

[11] Patent Number: 5,640,541

[45] **Date of Patent:** Jun. 17, 1997

- | | | | |
|-----------|---------|---------------|---------|
| 5,204,951 | 4/1993 | Keener et al. | 395/294 |
| 5,274,763 | 12/1993 | Banks | 395/250 |
| 5,430,849 | 7/1995 | Banks | 395/308 |

- OpenConnect Systems, *Distributed Solutions*, 1995.
OpenConnect Systems, *Internetwork Servers*, 1995.

- Primary Examiner—Kevin J. Teska**
Assistant Examiner—Dan Fiul
Attorney, Agent, or Firm—Baker & Botts, L.L.P.

- [57]
- ABSTRACT**

- An information system (10) including a plurality of computer systems is provided. The information system (10) includes a first computer system (16) having an IBM System/360/370 I/O interface channel (18). The first computer system (16) is operable to communicate SNA and non-SNA protocol information via the IBM System/360/370 I/O interface channel (18). The information system (10) includes a second computer system (40) having a SCSI bus (38). The second computer system (40) is operable to communicate SCSI protocol information via the SCSI bus (38). An adapter (36) is coupled to the IBM System/360/370 I/O interface channel (18) of the first computer system (16) and the SCSI bus (38) of the second computer system (40). The adapter (36) is operable to interface the SCSI bus (38) with the IBM System/360/370 I/O interface channel (18) to allow bi-directional communication between the first computer system (16) and the second computer system (40).

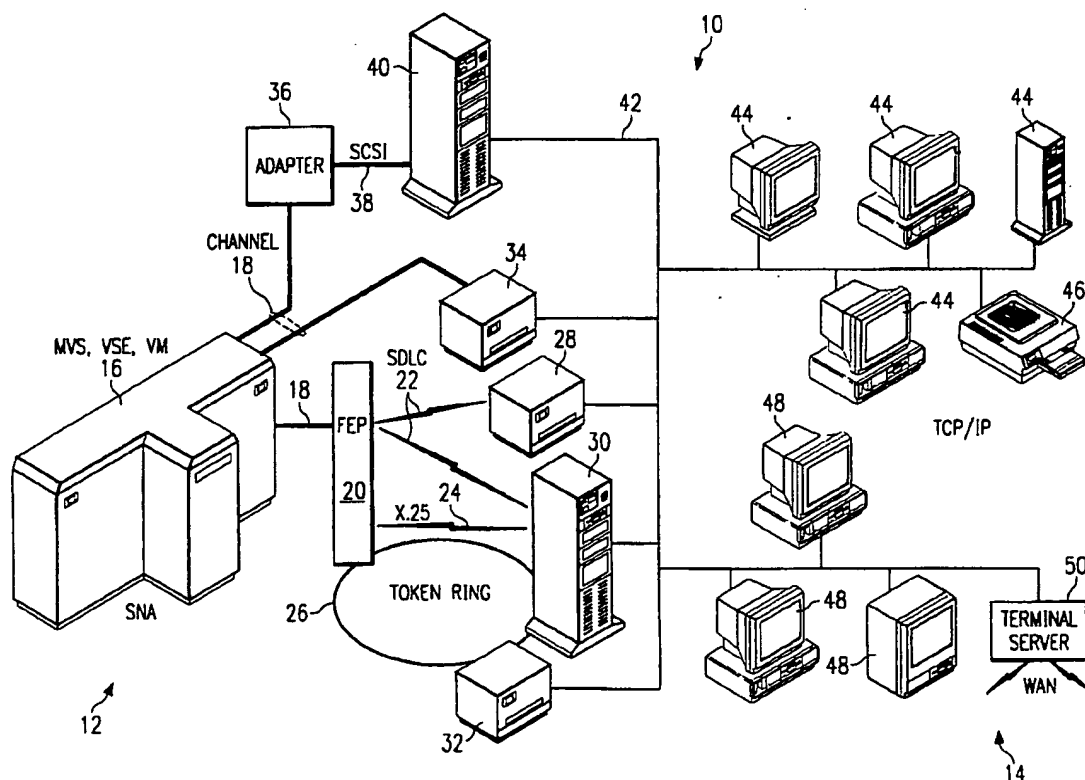
- [52] U.S. Cl. 395/500

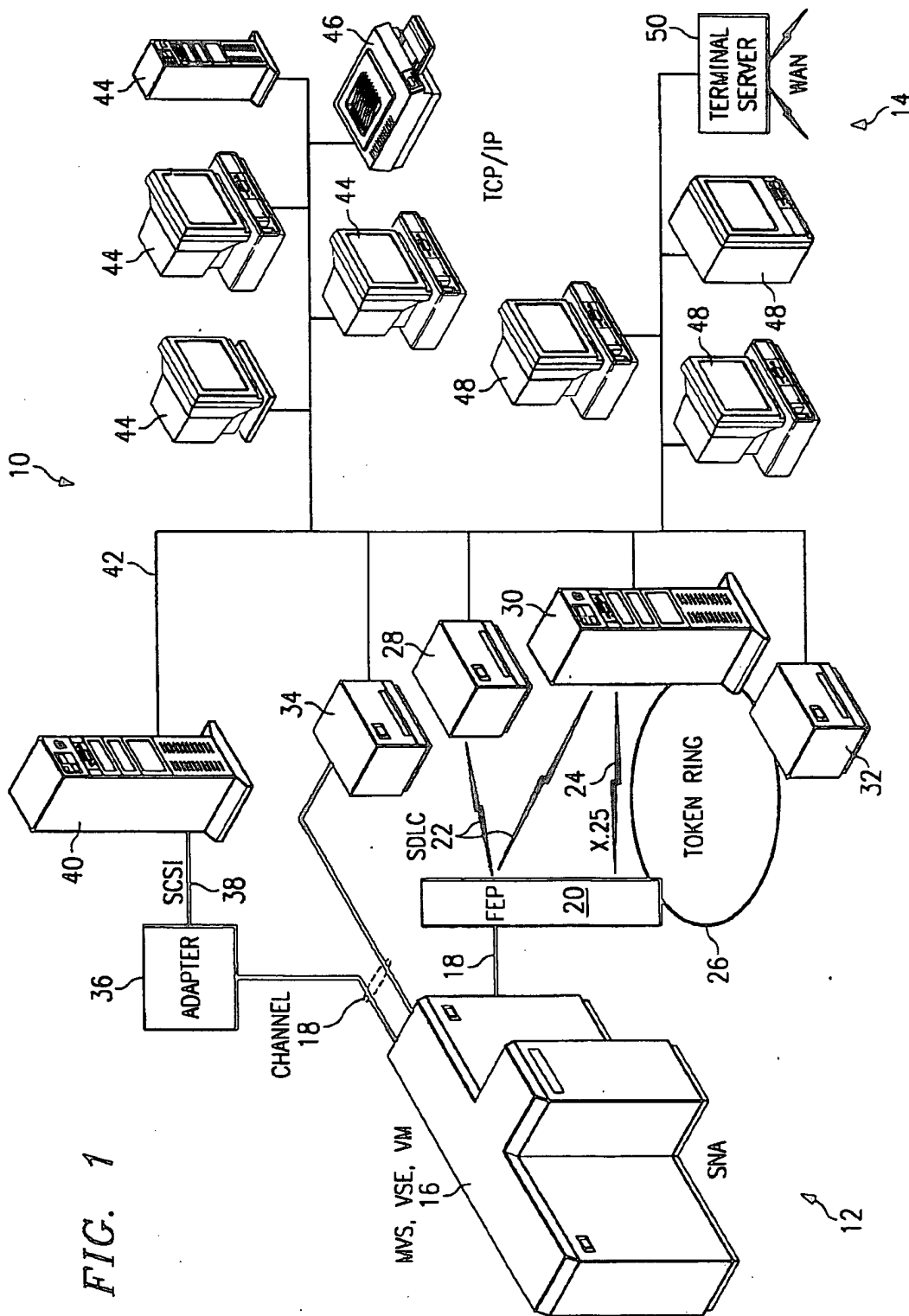
- [58] **Field of Search** 345/500, 309,
345/308, 294, 250, 253

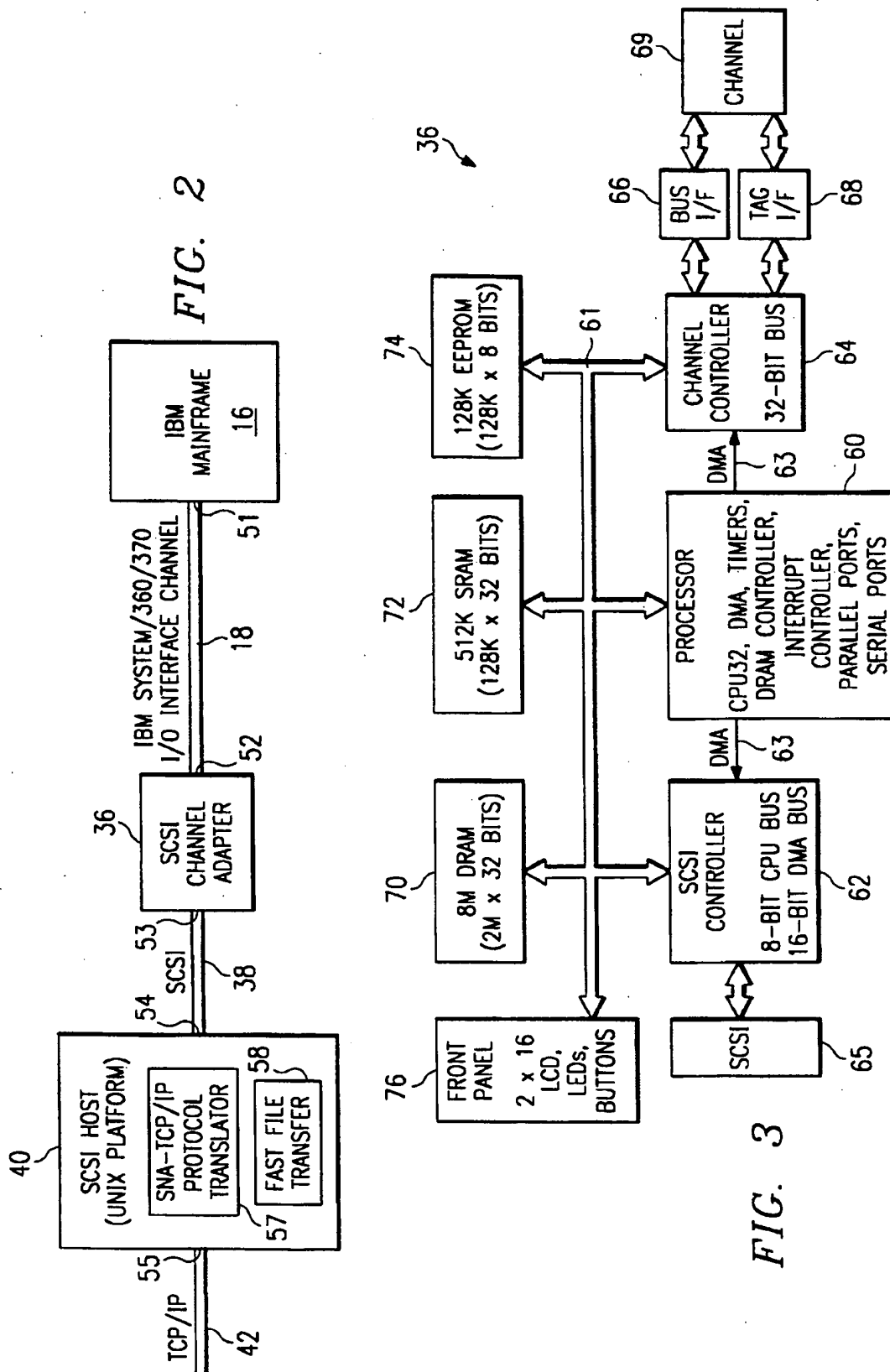
- U.S. PATENT DOCUMENTS

- | | | | |
|-----------|--------|----------------|---------|
| 5,113,500 | 5/1992 | Talbott et al. | 395/309 |
| 5,191,653 | 3/1993 | Banks et al. | 395/253 |

28 Claims, 54 Drawing Sheets







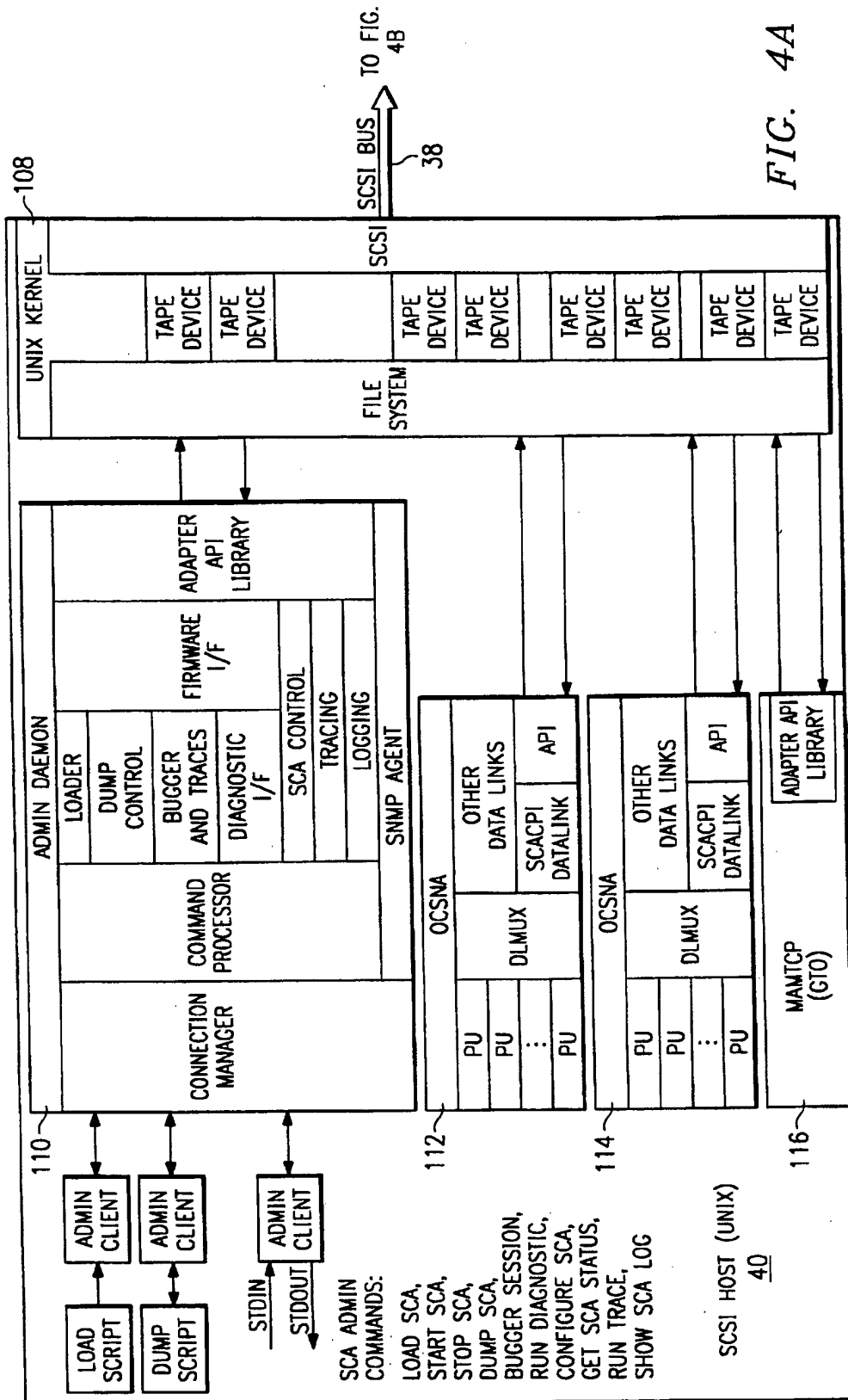
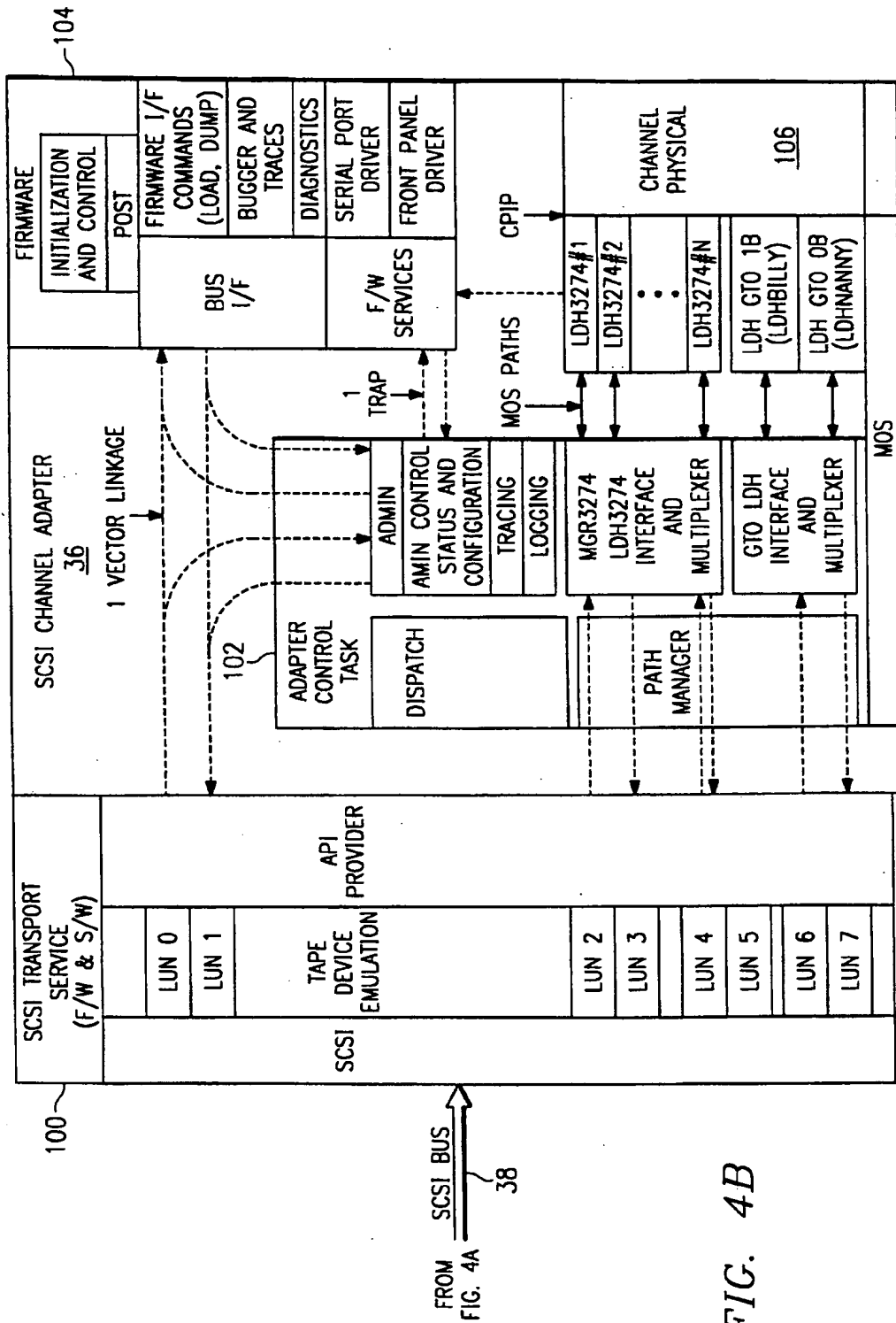


FIG. 4A



REFERENCE DOCUMENTATION

REF	PART	PAGE
U1	MC75128	6
U2	MC3485	8
U3	DS1231	8
U4	74F08	8,10
U5	MC3485	7
U6	29C828	8
U7	MAX233	2
U8	53C96	1
U9	74F14	4,6,8,10
U10	74F280A	7
U11	MC3485	7
U12	74HCT244	5
U13	29C621	7
U14	MC75128	8
U15	MC75128	7
U16	22V10	8
U17	XC4006	6
U18	25Mhz	2
U19	MC68360	2
U20	29C828	7
U21	MC3485	8
U22	74F163	8
U23	74F74	5,10
U24	74F280A	7
U25	MC3461	8
U26	74F163	8
U27	CY7C122	6
U28	18Mhz	8
U29	M8841000	3
U30	M8841000	3
U31	74A520	5
U32	74F138	2
U33	74F163	5
U34	UON3611	8
U35	22V10	5
U36	28F010	3
U37	74A6T245	3
U38	M8841000	3
U39	M8841000	3
U40	16M36	3
U41	74HCT123	5,10

DEVICE				PART NUMBER	DESCRIPTION
REF	TYPE	NUMBER	LABEL/FILE NAME		
				600-0060-101	ROM, ASSEMBLY
				600-0060-201	ASSEMBLY SPECIFICATION
				600-0060-301	SCHEMATIC DIAGRAM
				600-0060-401	TEST SPECIFICATION
				600-0060-402	TEST SOFTWARE
				600-0060-501	FABRICATION DRAWING
				600-0060-60X	ARTWORK
				600-0060-70X	SILKSCREEN
				600-0060-80X	SOLDERMASK, SOLDERPASTE
				600-0060-901	RAV PC BOARD
U35	GAL	22V10	0111R(+ REV)	575-0111-101	LCD CONTROLLER
U18	GAL	22V10	0112R(+ REV)	575-0112-101	RESET AND CHANNEL PRIORITY
U17	XILINX	XC4006-8	0017A	270-0017-301	CHANNEL SCHEMATIC 4006

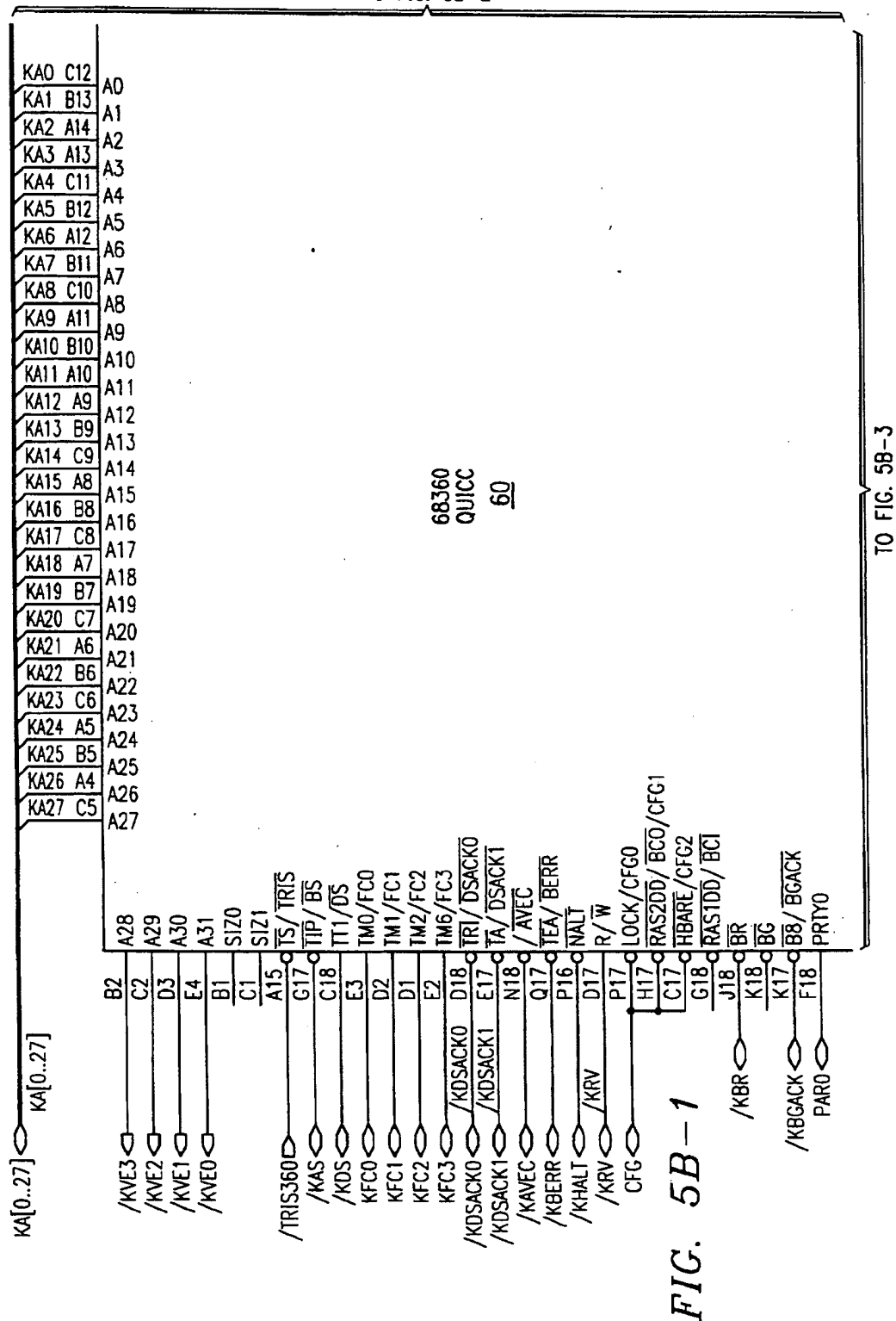
NOTES: UNLESS OTHERWISE SPECIFIED

- 1) ALL PULLUP RESISTORS ARE 10K OHM SURFACE MOUNT 16 PIN 50 TYPE AND ARE ON PAGE 10.
- 2) ALL CAPACITORS ARE .1uf.
- 3) ALL SPARE PARTS ARE ON PAGE 10.

| LINK
 | PG2.sch
 | PG3.sch
 | PG4.sch
 | PG5.sch
 | PG6.sch
 | PG7.sch
 | PG8.sch
 | PG9.sch
 | PG10.sch
 | PG11.sch

FIG. 5A

TO FIG. 5B-2



TO FIG. 5B-3

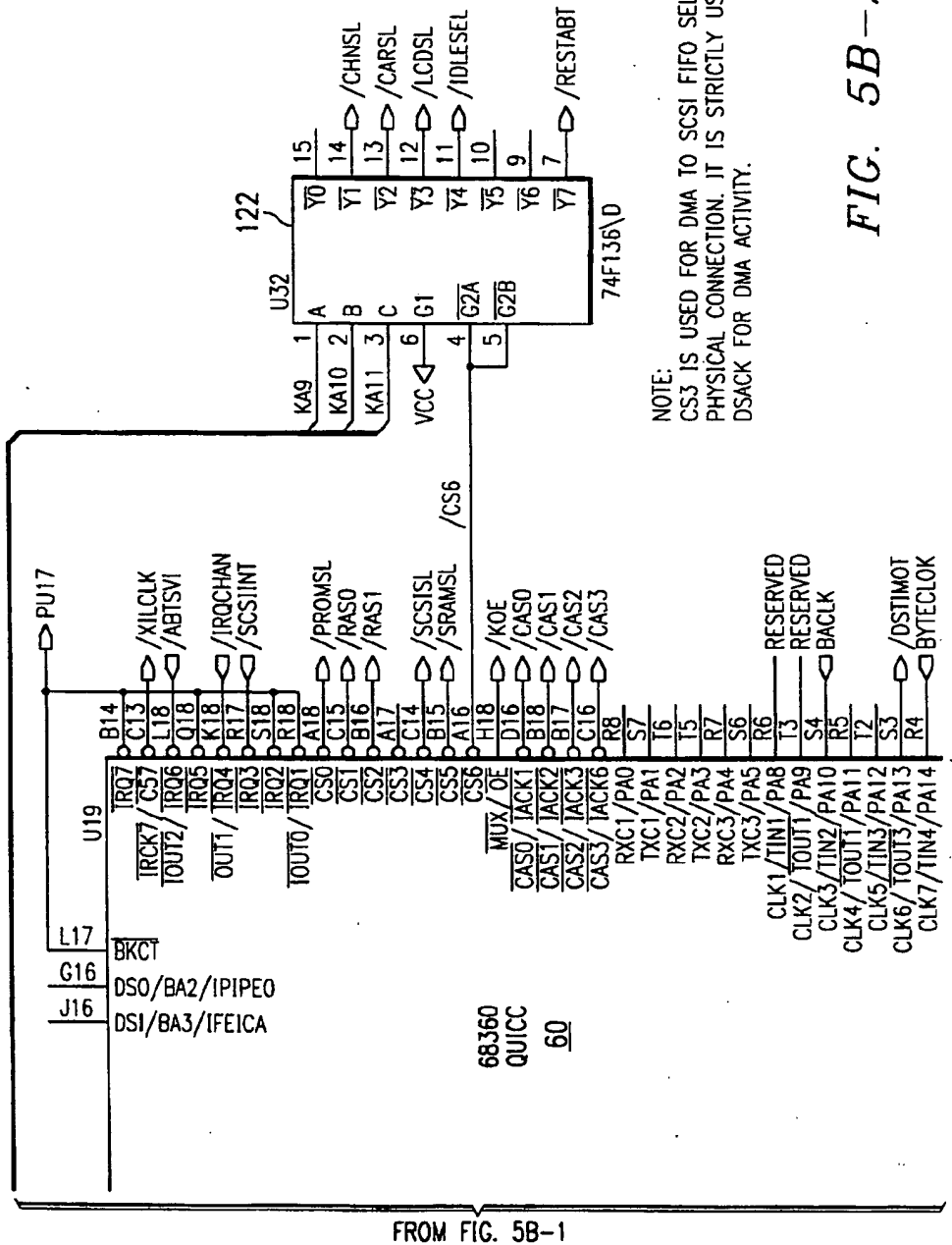


FIG. 5B-2

TO FIG. 5B-4

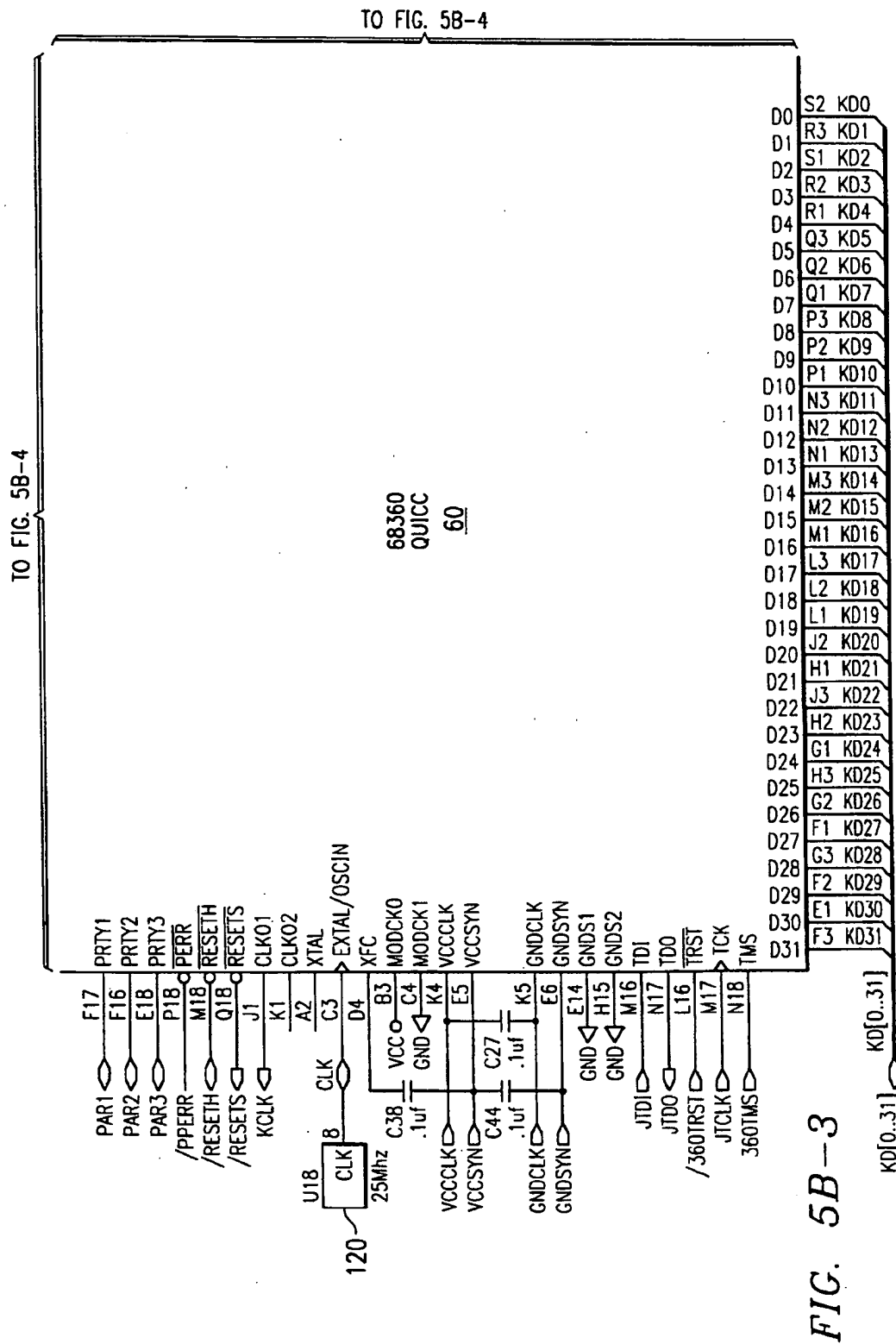
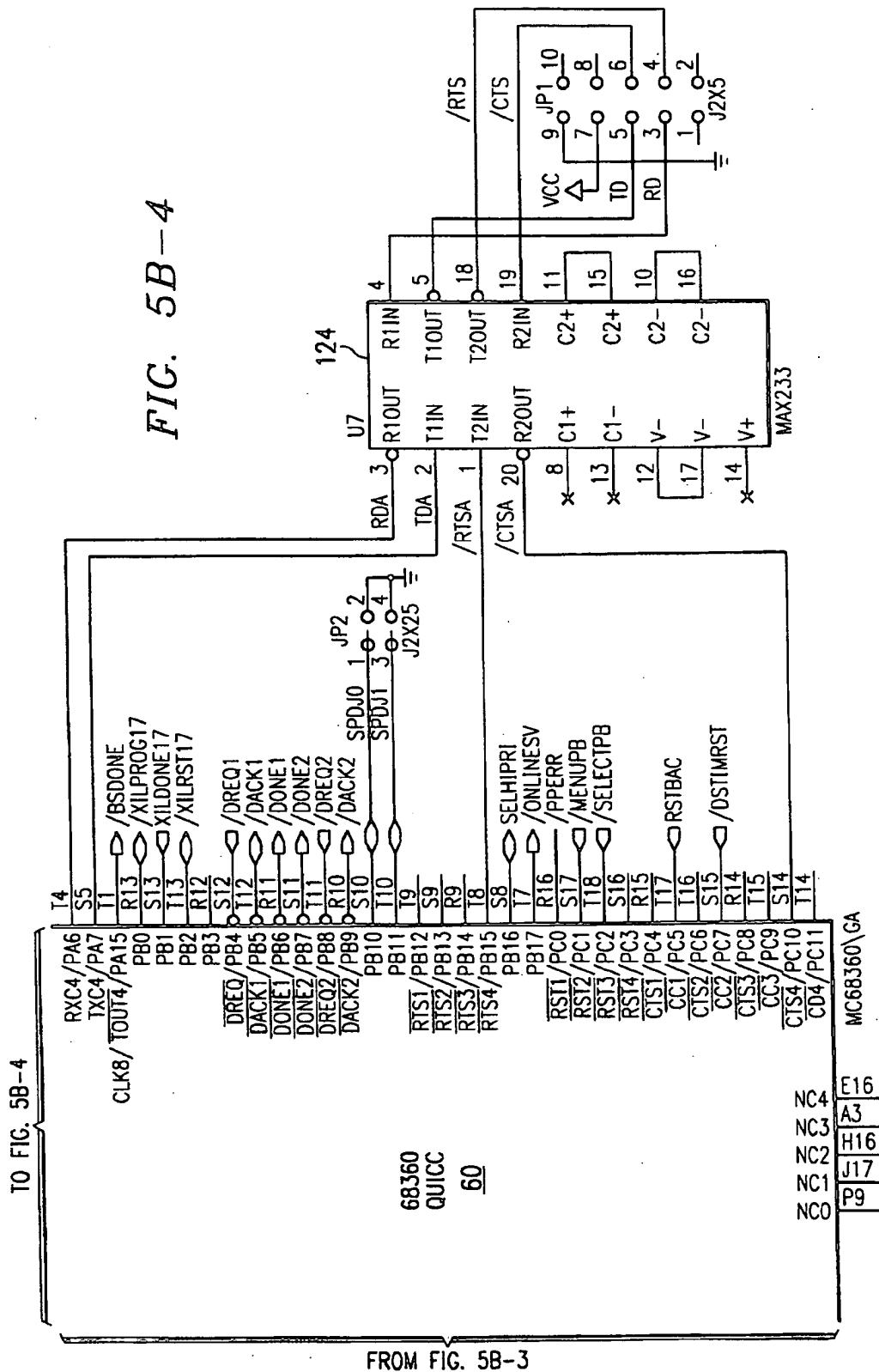
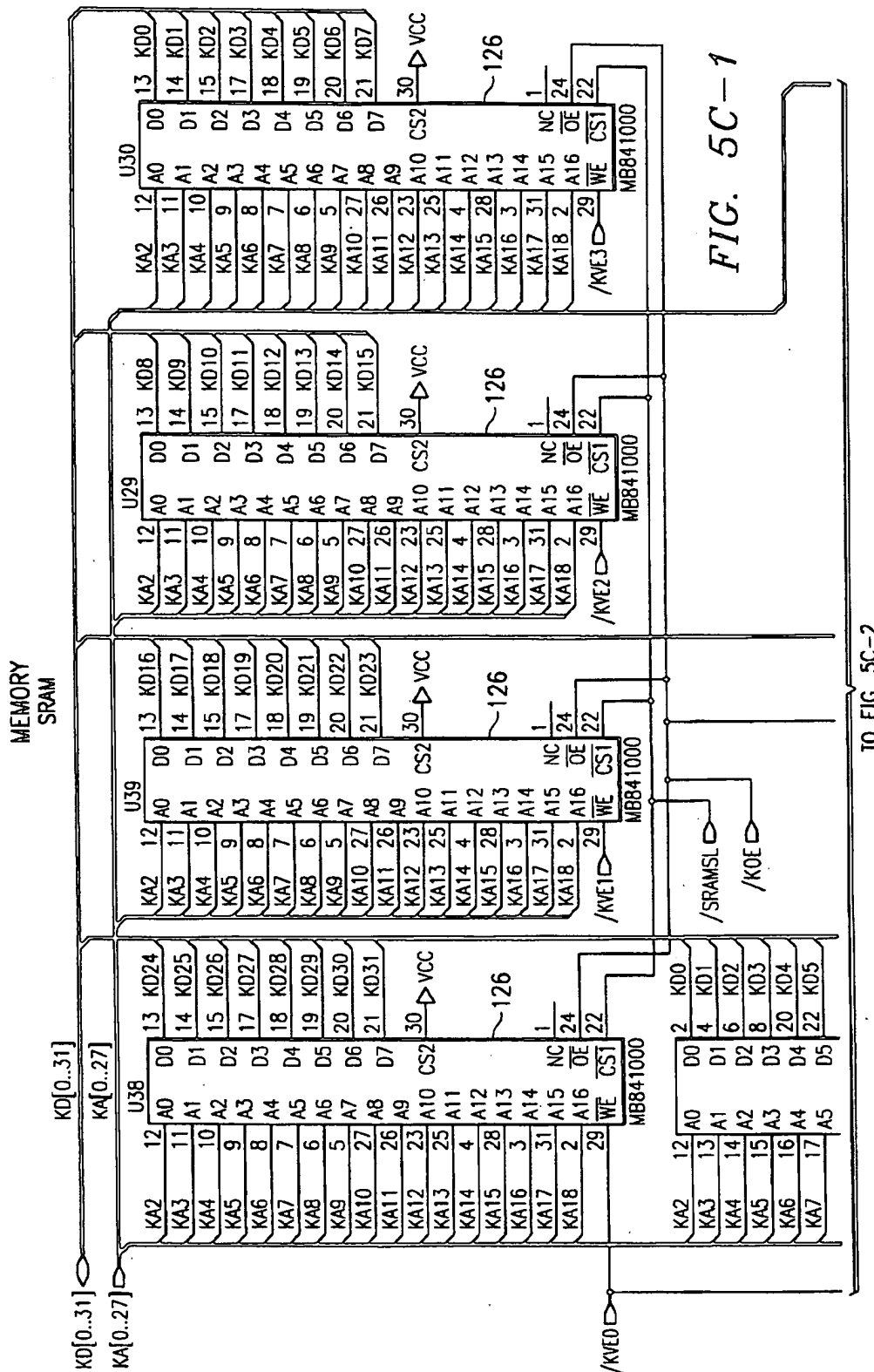


FIG. 5B-4





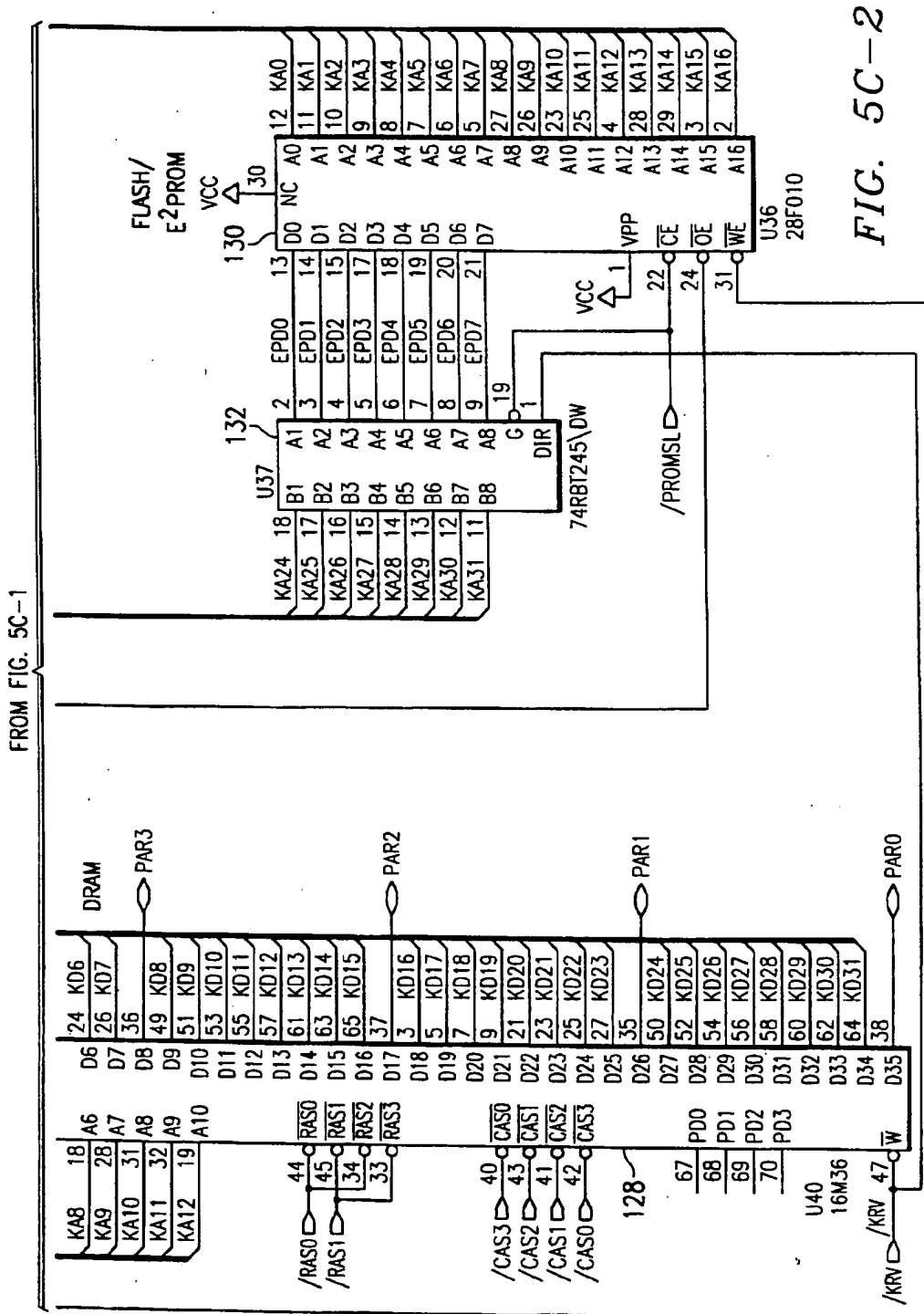


FIG. 5C-2

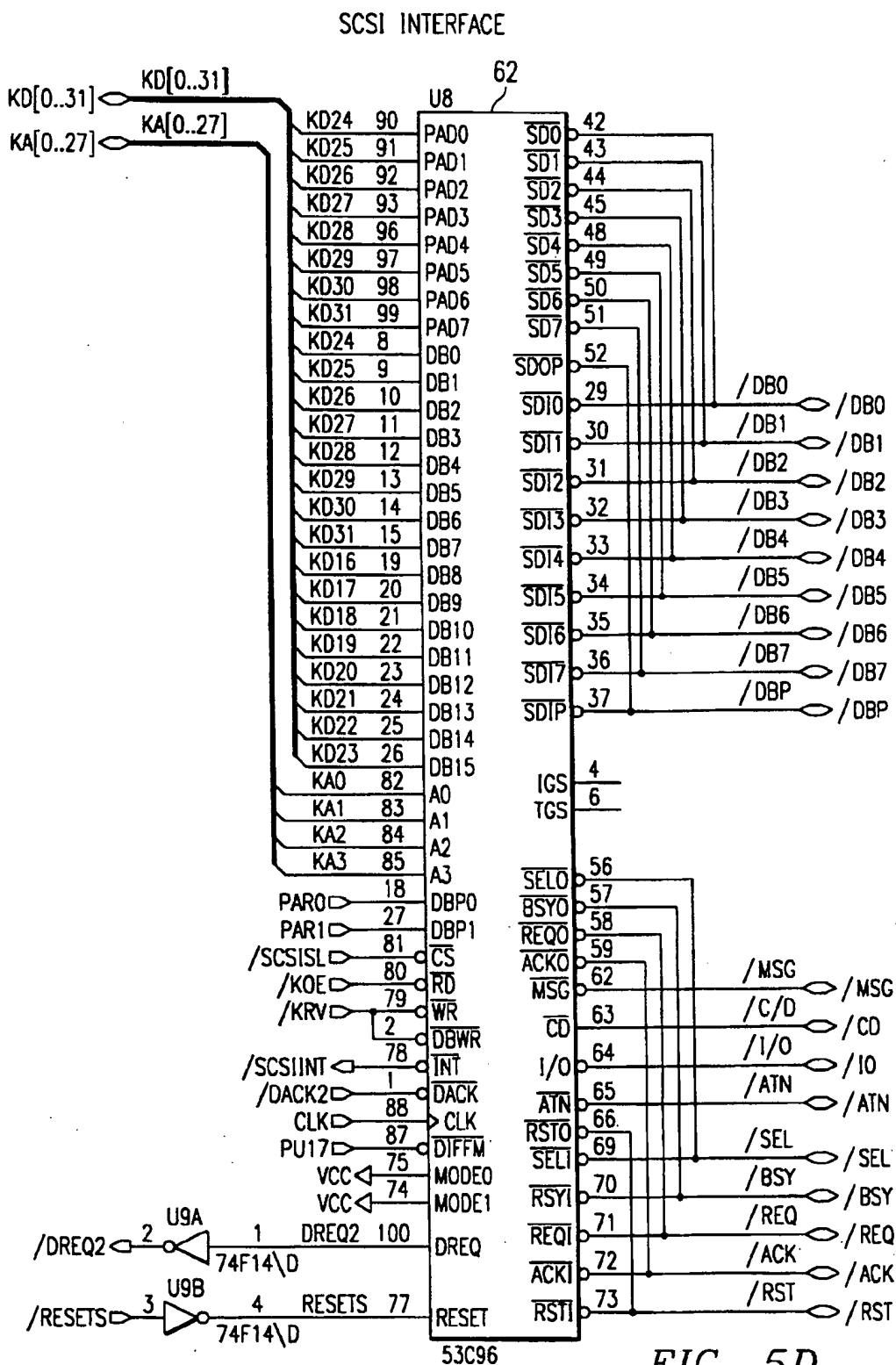
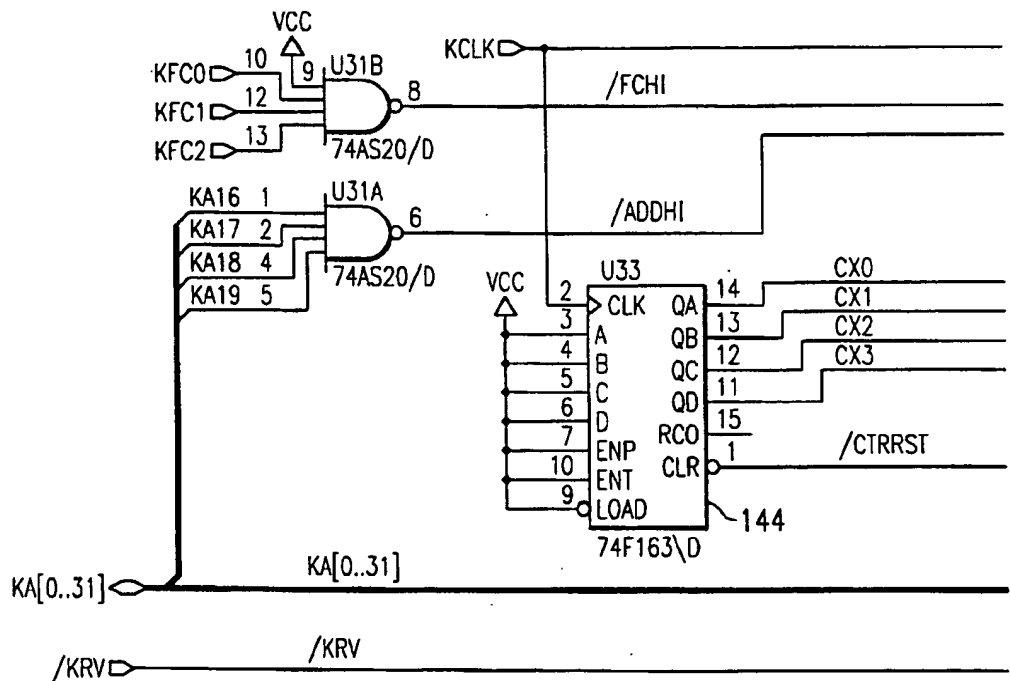
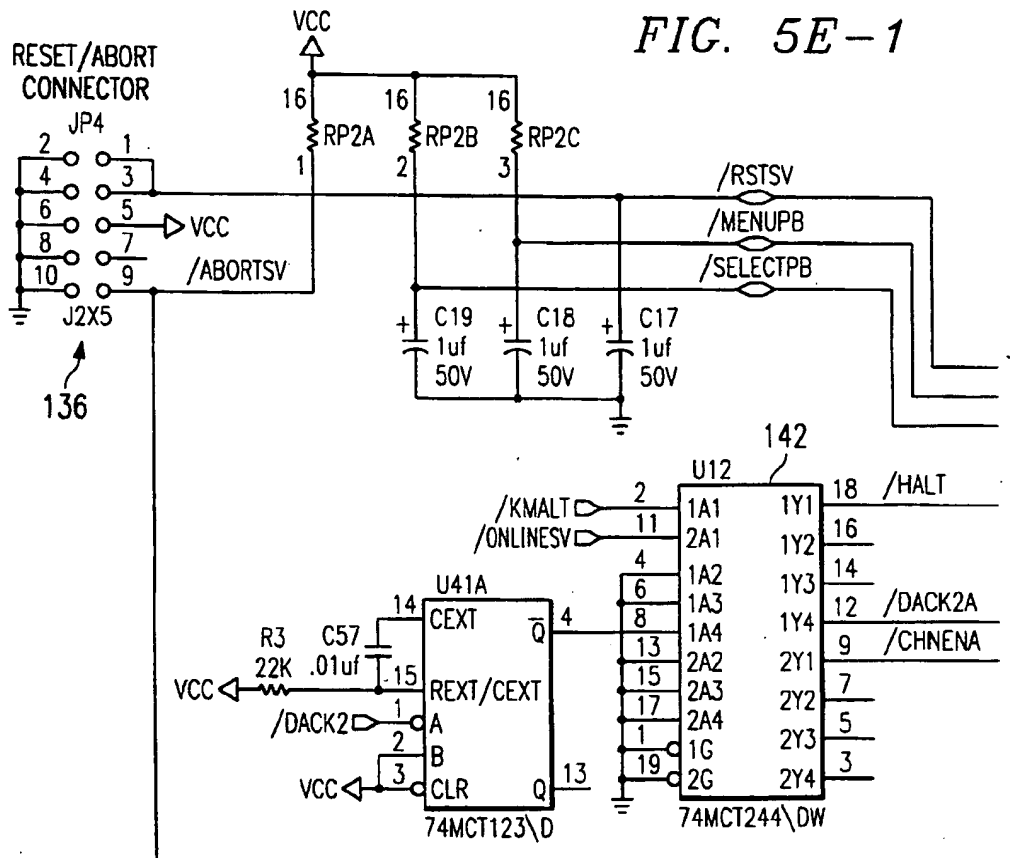
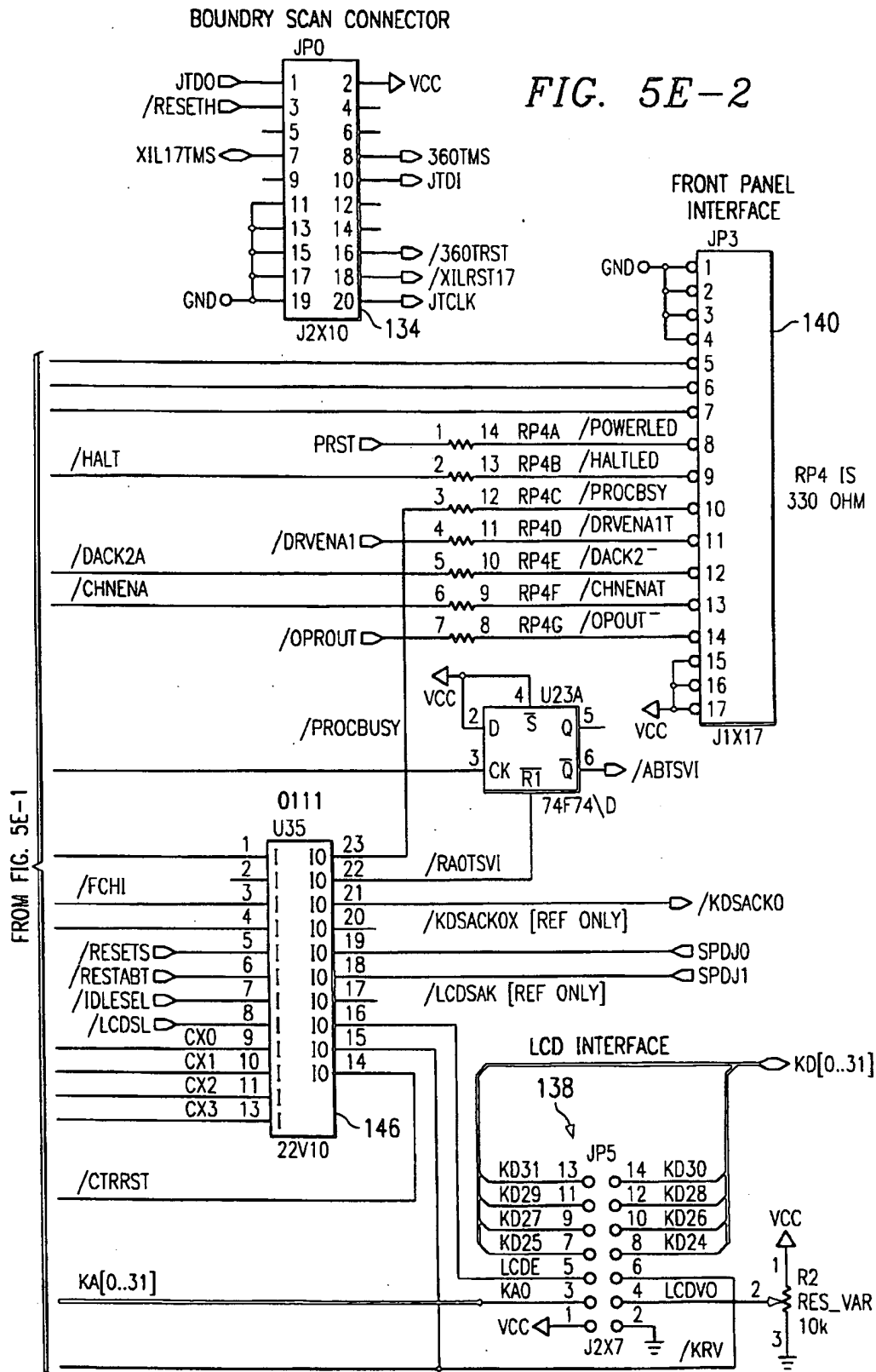


FIG. 5D

FIG. 5E-1



TO FIG. 5E-2



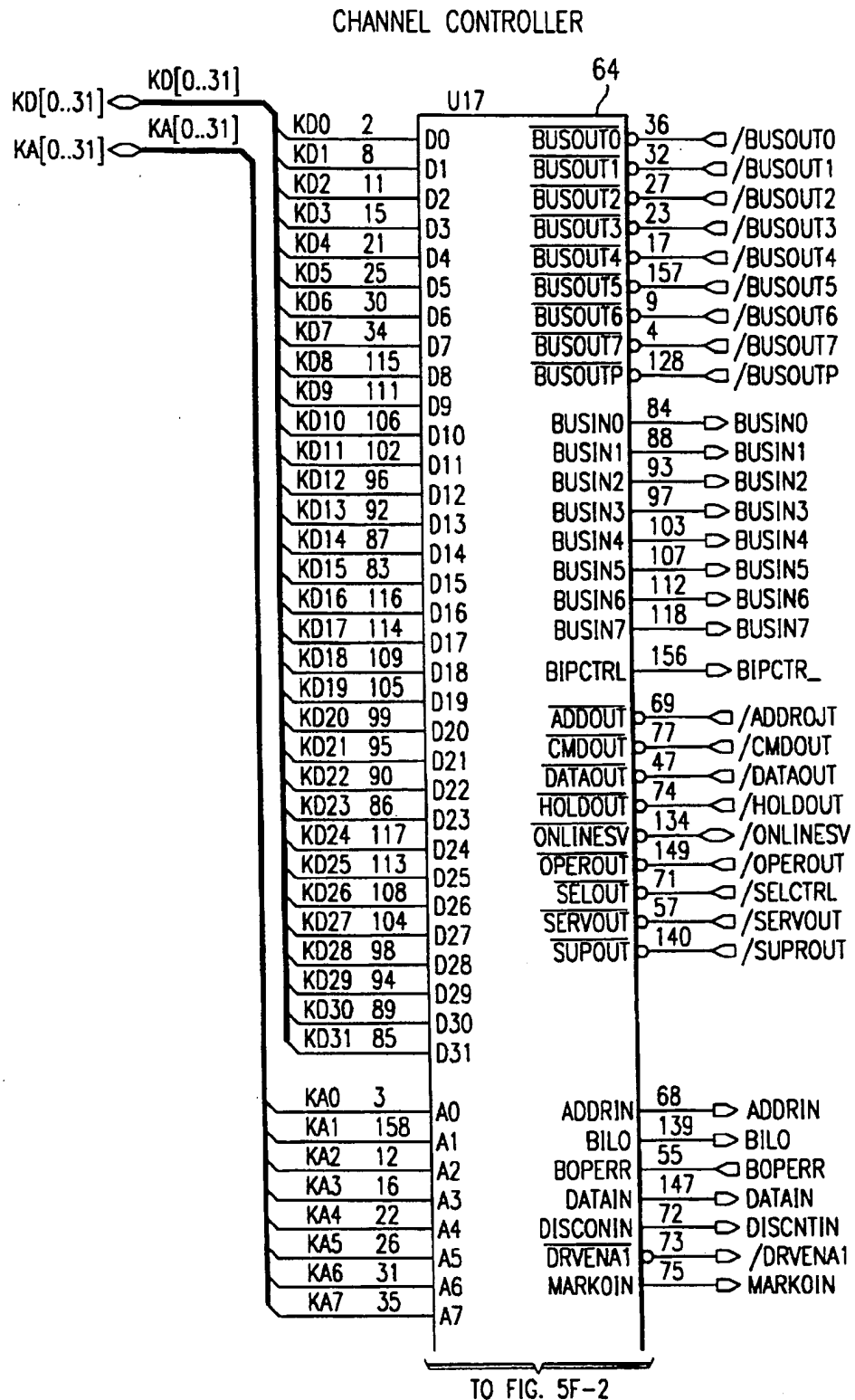


FIG. 5F-1

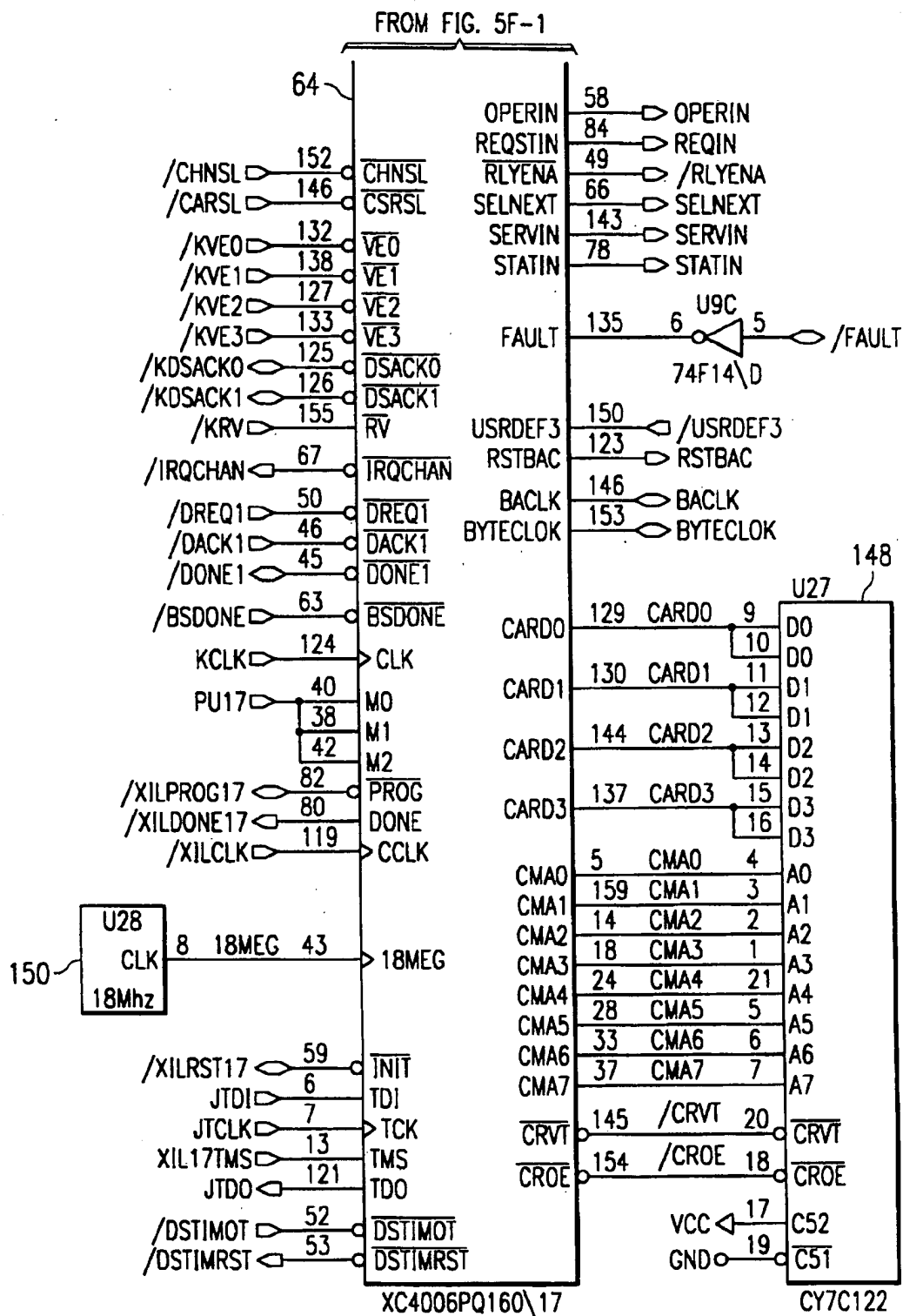
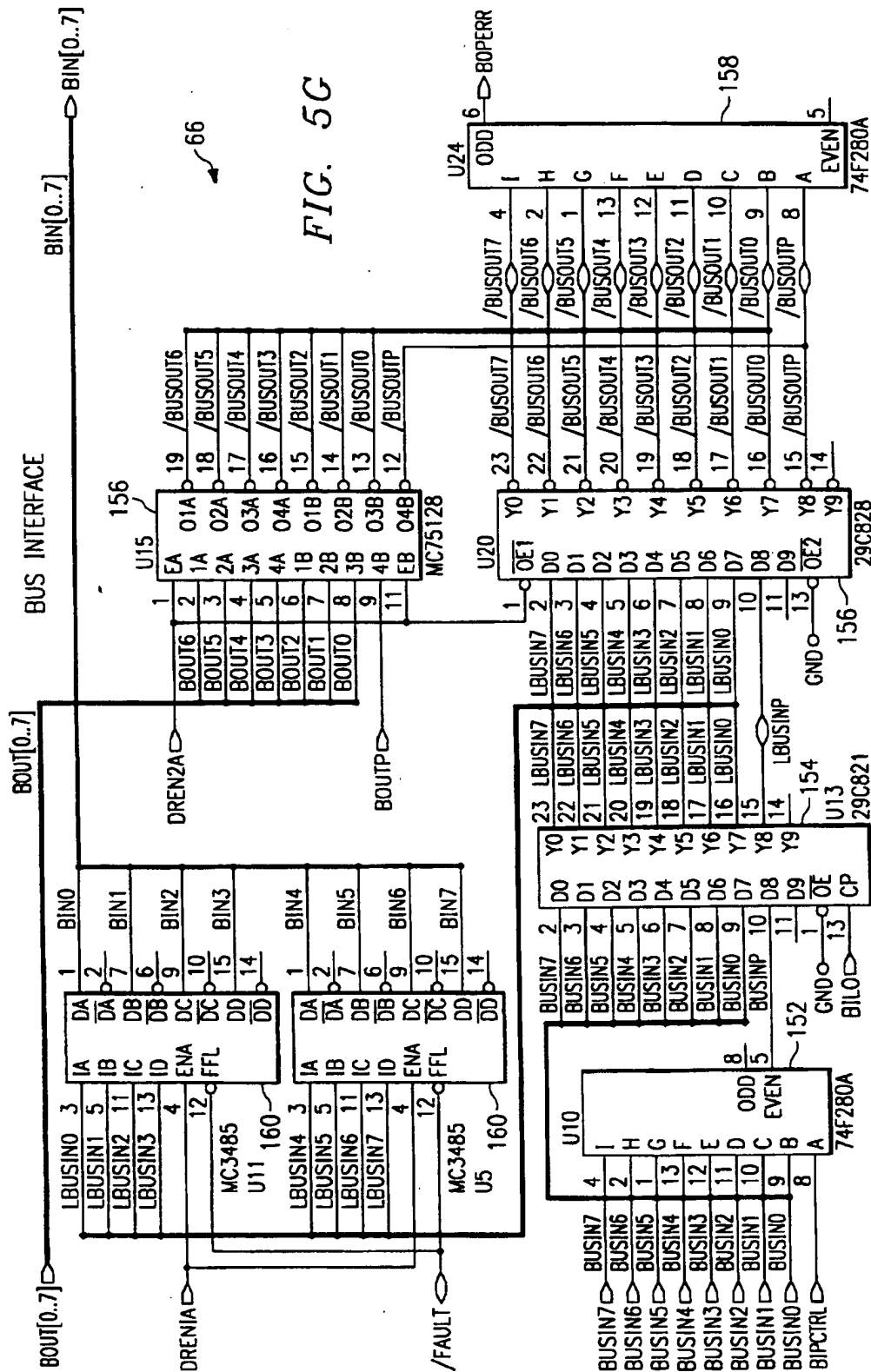


FIG. 5F-2



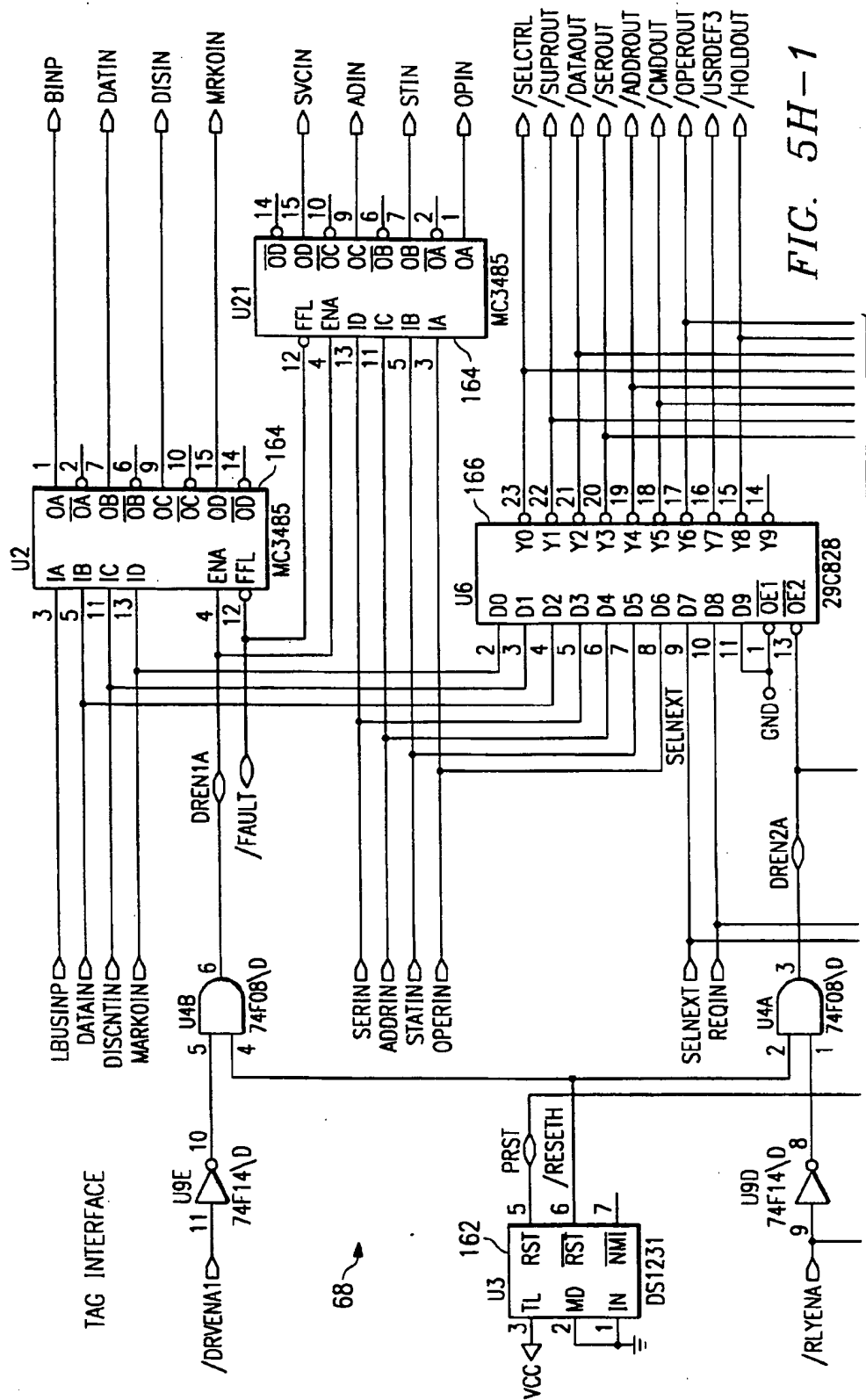
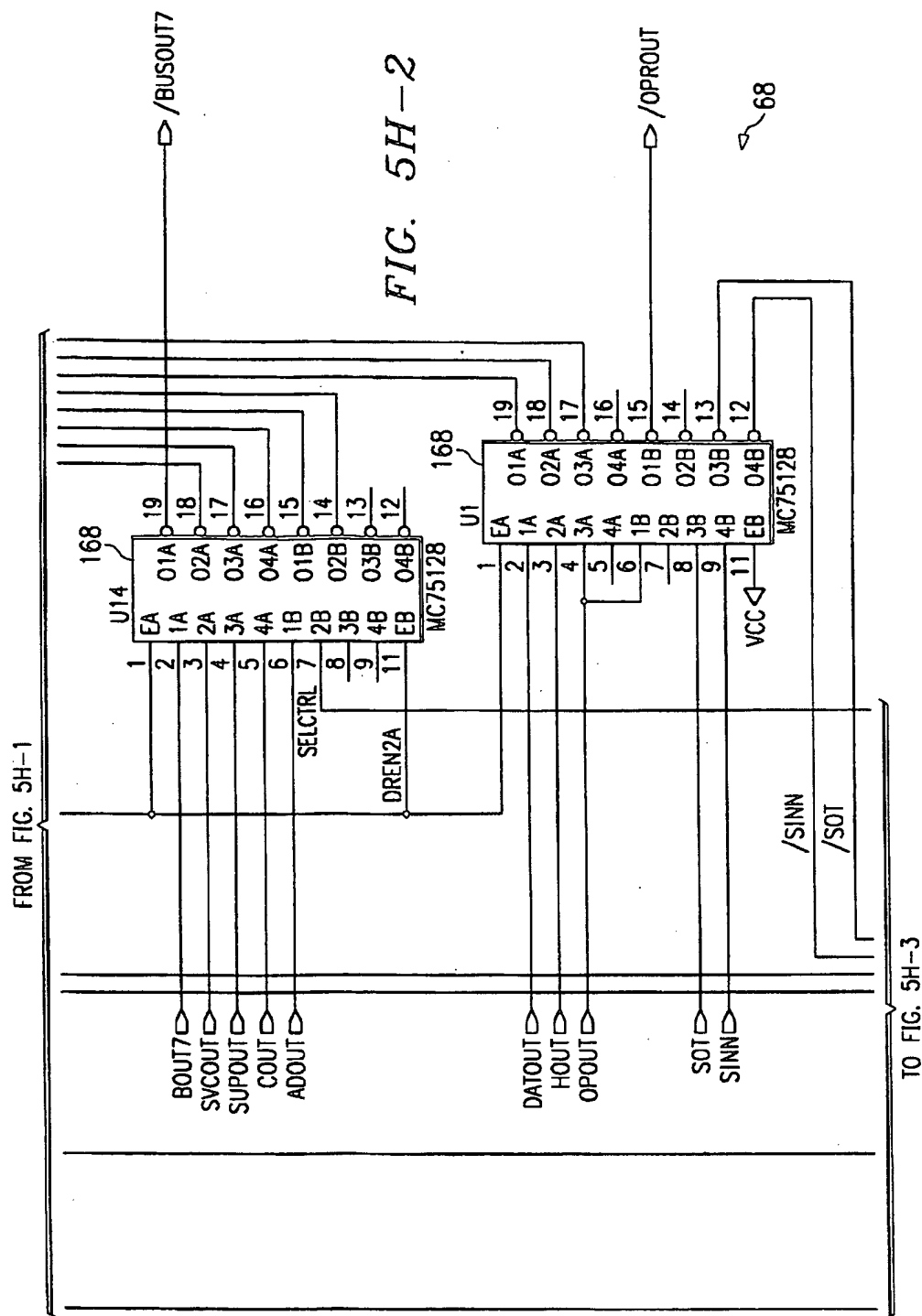


FIG. 5H-1

TO FIG. 5H-2



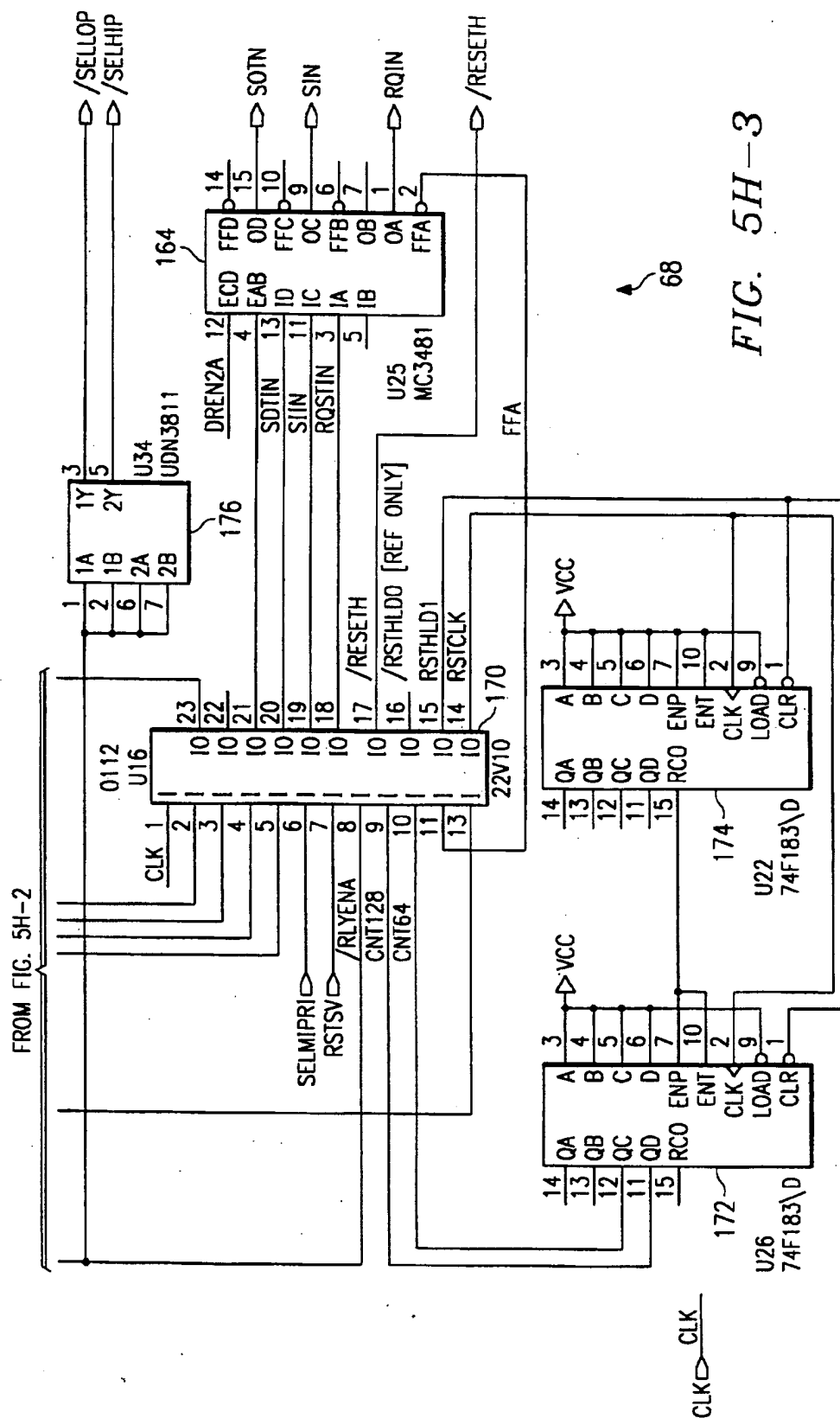


FIG. 5H-3

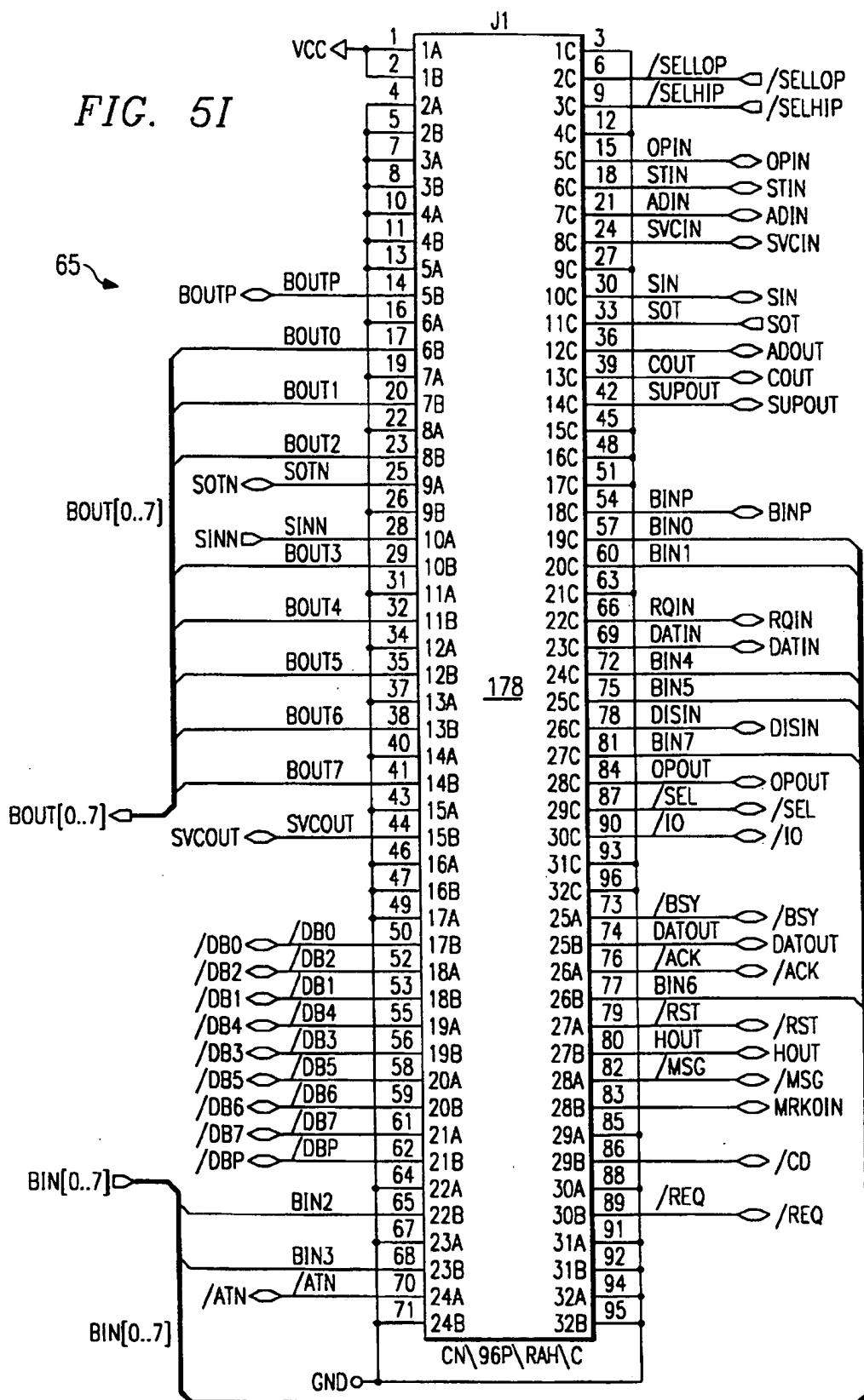
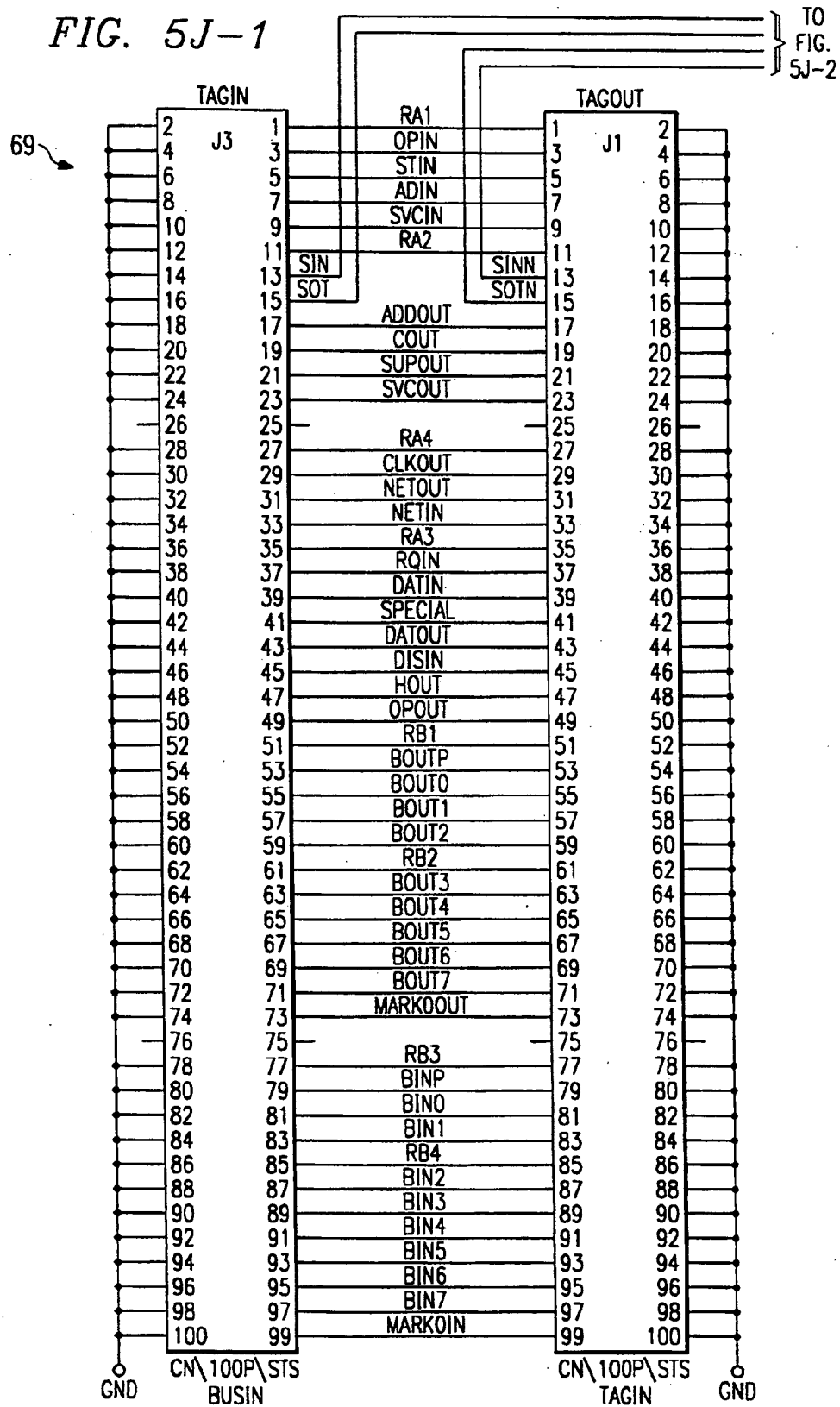
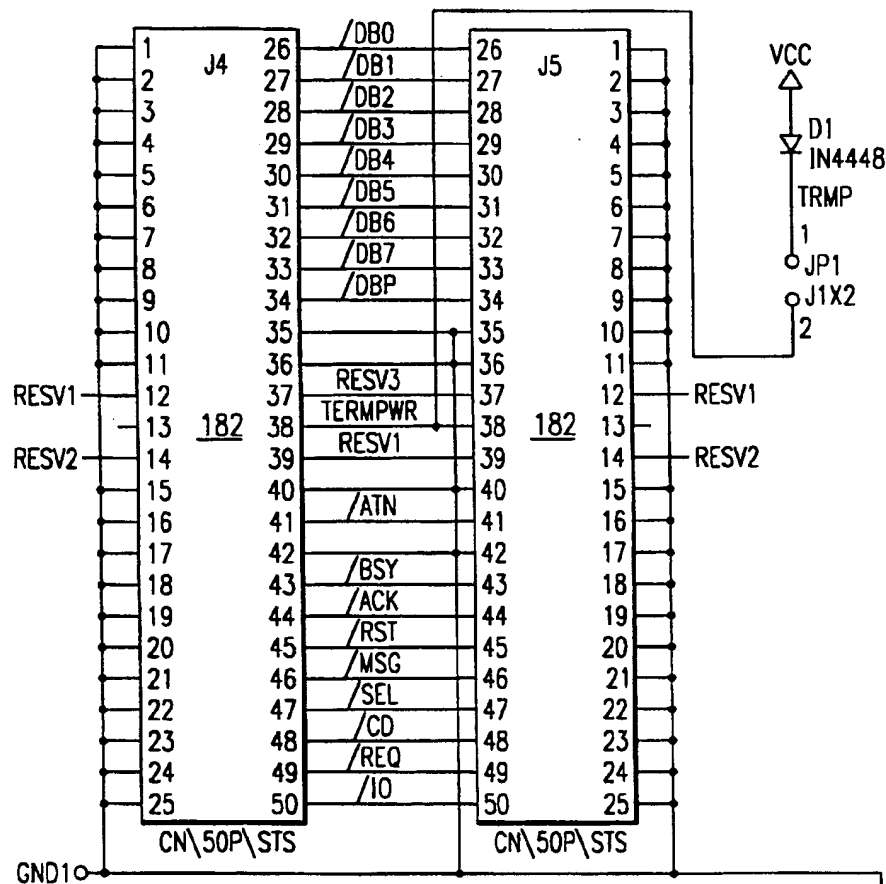
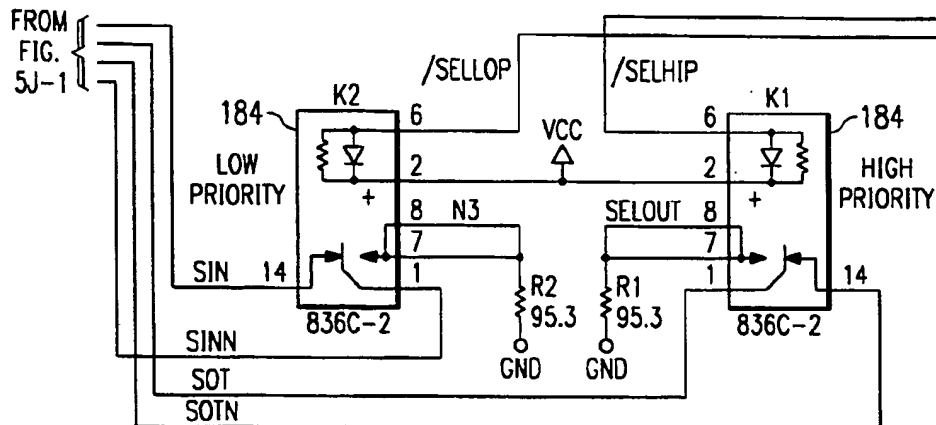


FIG. 5J-1





NOTE:
RELAYS SHOWN DISABLED.

REFERENCE
DOCUMENTION

FIG. 5J-2

PART NUMBER	DESCRIPTION
600-0067-101	ROM. ASSEMBLY
600-0067-201	ASSEMBLY SPECIFICATION
600-0067-301	SCHEMATIC DIAGRAM
600-0067-401	TEST SPECIFICATION
600-0067-402	TEST SOFTWARE
600-0067-501	FABRICATION DRAWING
600-0067-60X	ARTWORK
600-0067-70X	SILKSCREEN

TO FIG. 5J-3

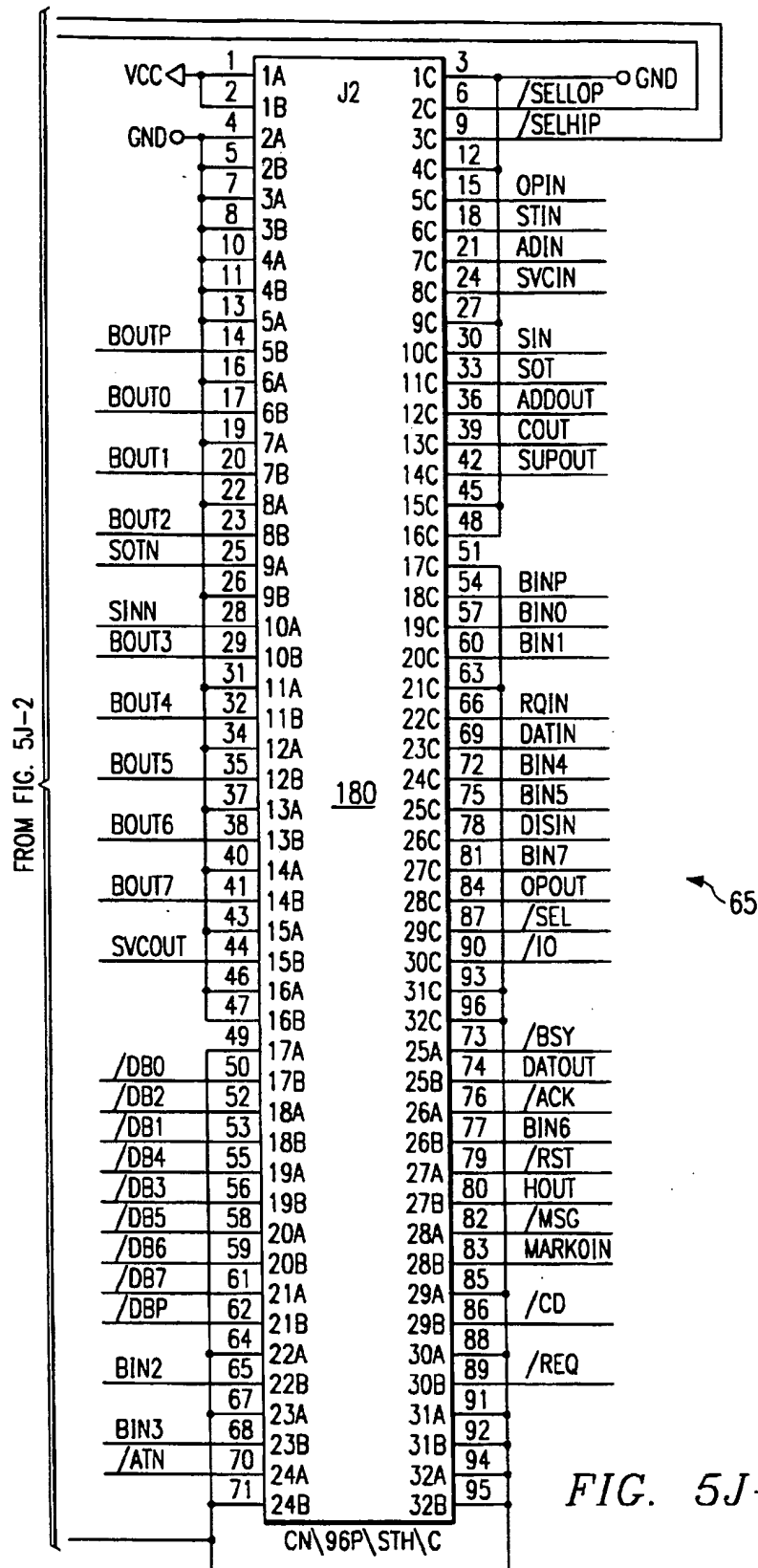


FIG. 5J-3

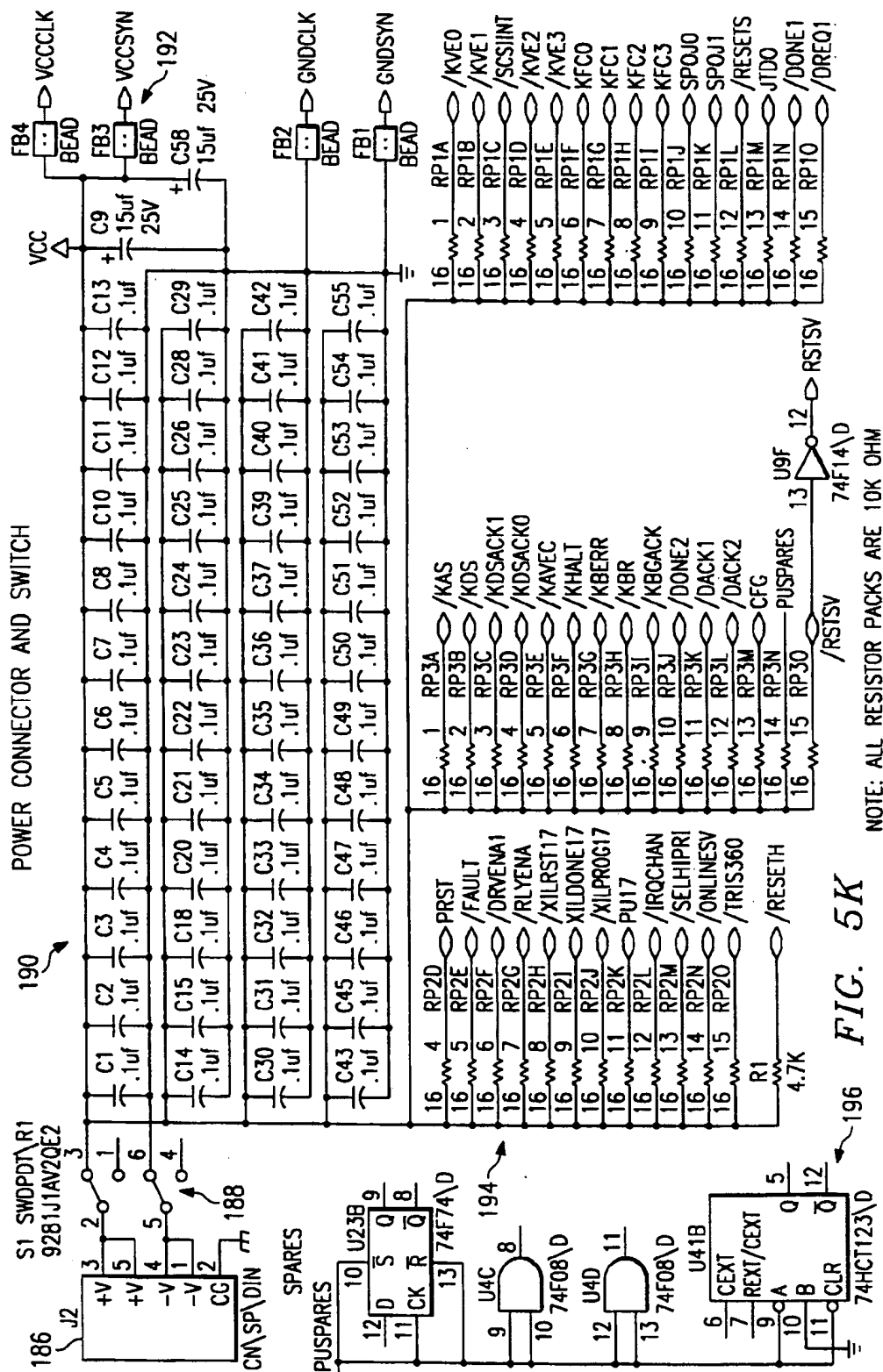
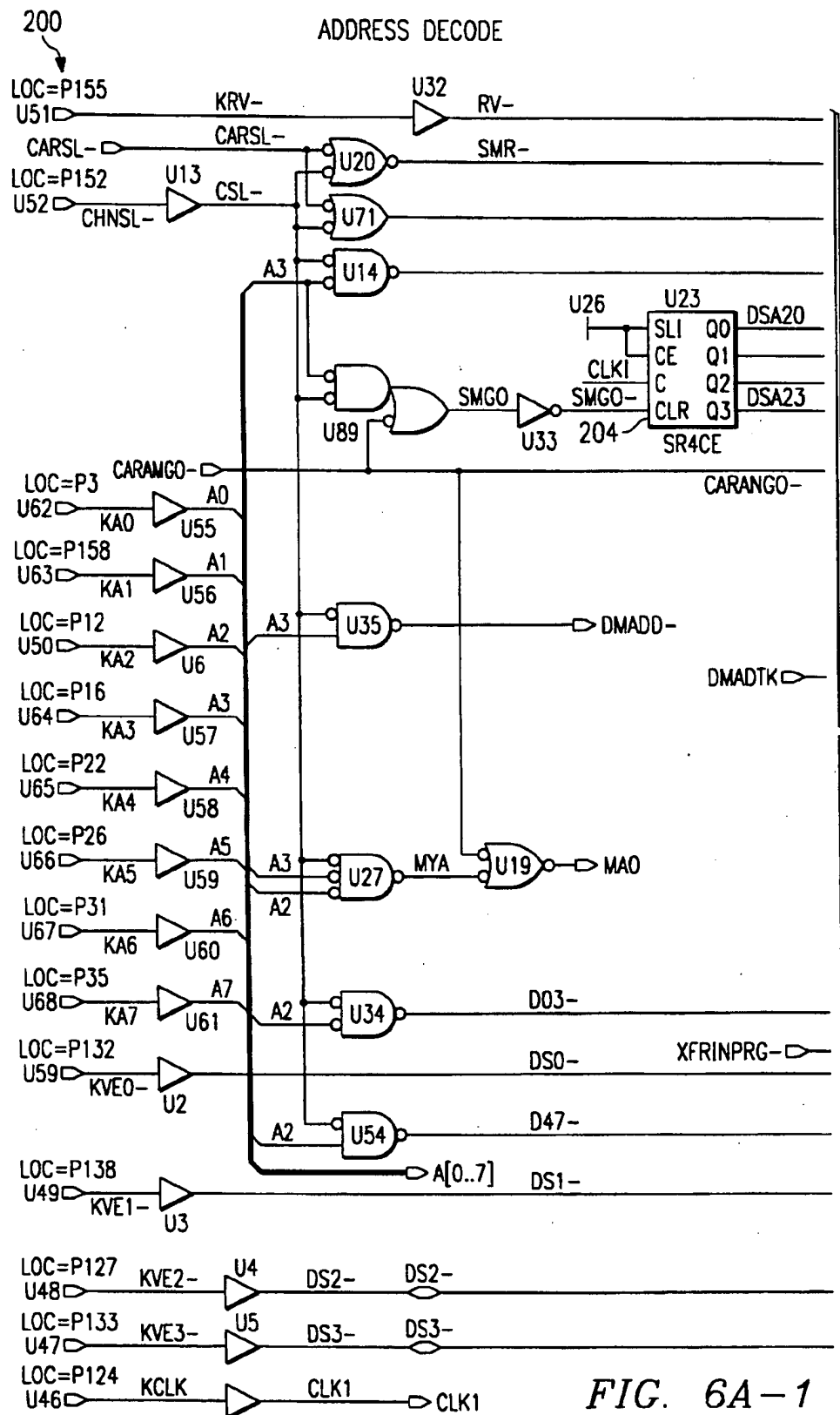


FIG. 5K

NOTE: ALL RESISTOR PACKS ARE 10K OHM



ADDRESS DECODE

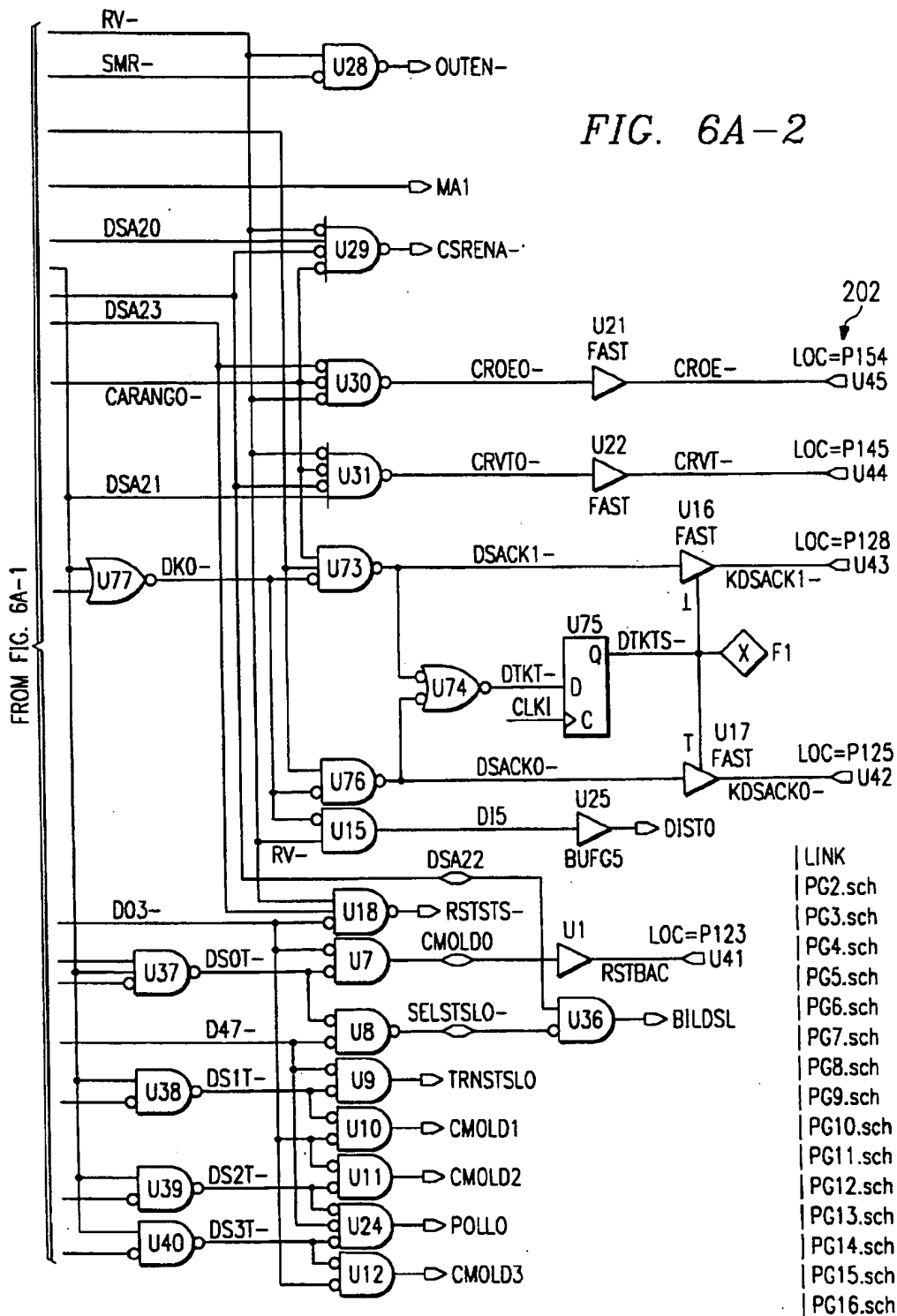


FIG. 6B

COMMAND REGISTER 0 D[0..31]

COMMAND REGISTER 1

COMMAND REGISTER 2

COMMAND REGISTER 3

CMOLD0

XFC2

WT

F2

RD

ROW

214

CUBUSY

TERMINATE

FORCLOK

216

FCEB

ABTPOLL

SUPABLS

DISCONIN

LOC=P72

U112

LOC=P75

U111

MARKOIN

U78

MOI

U100

NDMOT

D0

MRKIN

D1

S0

DIAGNOSTIC REGISTER 3

U85

U87

U88

U89

D0

D1

D2

D3

D4

D8

D7

DIAG-

206

DIAG

U98

DIAG

U97

MAP=PUD

MOI

MST

MRKIN

DIAG

LOC=P156

U110

BIPCTRL

U80

BIPCD

DAI

DOI

DRI

DSI

DDI

DSRI

208

CMOLD2

CMOLD3

CMOLD1

TSMSMPT

DIAG

F3

U96

U93

U94

U95

D8

D11

D14

D15

212

INTENA

210

U82

U83

LOOPBACK

U81

RLYENA-

LOC=P49

U109

208

CMOLD2

CMOLD3

CMOLD1

TSMSMPT

DIAG

F3

U96

U93

U94

U95

D8

D11

D14

D15

212

INTENA

210

U82

U83

LOOPBACK

U81

RLYENA-

LOC=P49

U109

208

CMOLD2

CMOLD3

CMOLD1

TSMSMPT

DIAG

F3

U96

U93

U94

U95

D8

D11

D14

D15

212

INTENA

210

U82

U83

LOOPBACK

U81

RLYENA-

LOC=P49

U109

208

CMOLD2

CMOLD3

CMOLD1

TSMSMPT

DIAG

F3

U96

U93

U94

U95

D8

D11

D14

D15

212

INTENA

210

U82

U83

LOOPBACK

U81

RLYENA-

LOC=P49

U109

208

CMOLD2

CMOLD3

CMOLD1

TSMSMPT

DIAG

F3

U96

U93

U94

U95

D8

D11

D14

D15

212

INTENA

210

U82

U83

LOOPBACK

U81

RLYENA-

LOC=P49

U109

208

CMOLD2

CMOLD3

CMOLD1

TSMSMPT

DIAG

F3

U96

U93

U94

U95

D8

D11

D14

D15

212

INTENA

210

U82

U83

LOOPBACK

U81

RLYENA-

LOC=P49

U109

208

CMOLD2

CMOLD3

CMOLD1

TSMSMPT

DIAG

F3

U96

U93

U94

U95

D8

D11

D14

D15

212

INTENA

210

U82

U83

LOOPBACK

U81

RLYENA-

LOC=P49

U109

208

CMOLD2

CMOLD3

CMOLD1

TSMSMPT

DIAG

F3

U96

U93

U94

U95

D8

D11

D14

D15

212

INTENA

210

U82

U83

LOOPBACK

U81

RLYENA-

LOC=P49

U109

208

CMOLD2

CMOLD3

CMOLD1

TSMSMPT

DIAG

F3

U96

U93

U94

U95

D8

D11

D14

D15

212

INTENA

210

U82

U83

LOOPBACK

U81

RLYENA-

LOC=P49

U109

208

CMOLD2

CMOLD3

CMOLD1

TSMSMPT

DIAG

F3

U96

U93

U94

U95

D8

D11

D14

D15

212

INTENA

210

U82

U83

LOOPBACK

U81

RLYENA-

LOC=P49

U109

208

CMOLD2

CMOLD3

CMOLD1

TSMSMPT

DIAG

F3

U96

U93

U94

U95

D8

D11

D14

D15

212

INTENA

210

U82

U83

LOOPBACK

U81

RLYENA-

LOC=P49

U109

208

CMOLD2

CMOLD3

CMOLD1

TSMSMPT

DIAG

F3

U96

U93

U94

U95

D8

D11

D14

D15

212

INTENA

210

U82

U83

LOOPBACK

U81

RLYENA-

LOC=P49

U109

208

CMOLD2

CMOLD3

CMOLD1

TSMSMPT

DIAG

F3

U96

U93

U94

U95

D8

D11

D14

D15

212

INTENA

210

U82

U83

LOOPBACK

U81

RLYENA-

LOC=P49

U109

208

CMOLD2

CMOLD3

CMOLD1

TSMSMPT

DIAG

F3

U96

U93

U94

U95

D8

D11

D14

D15

212

INTENA

210

U82

U83

LOOPBACK

U81

RLYENA-

LOC=P49

U109

208

CMOLD2

CMOLD3

CMOLD1

TSMSMPT

DIAG

F3

U96

U93

U94

U95

D8

D11

D14

D15

212

INTENA

210

U82

U83

LOOPBACK

U81

RLYENA-

LOC=P49

U109

208

CMOLD2

CMOLD3

CMOLD1

TSMSMPT

DIAG

F3

U96

U93

U94

U95

D8

D11

D14

D15

212

INTENA

210

U82

U83

LOOPBACK

U81

RLYENA-

LOC=P49

U109

208

CMOLD2

CMOLD3

CMOLD1

TSMSMPT

DIAG

F3

U96

U93

U94

U95

D8

D11

D14

D15

212

INTENA

210

U82

U83

LOOPBACK

U81

RLYENA-

LOC=P49

U109

208

CMOLD2

CMOLD3

CMOLD1

TSMSMPT

DIAG

F3

U96

U93

U94

U95

D8

D11

D14

D15

212

INTENA

210

U82

U83

LOOPBACK

U81

RLYENA-

LOC=P49

U109

208

CMOLD2

CMOLD3

CMOLD1

TSMSMPT

DIAG

F3

U96

U93

U94

U95

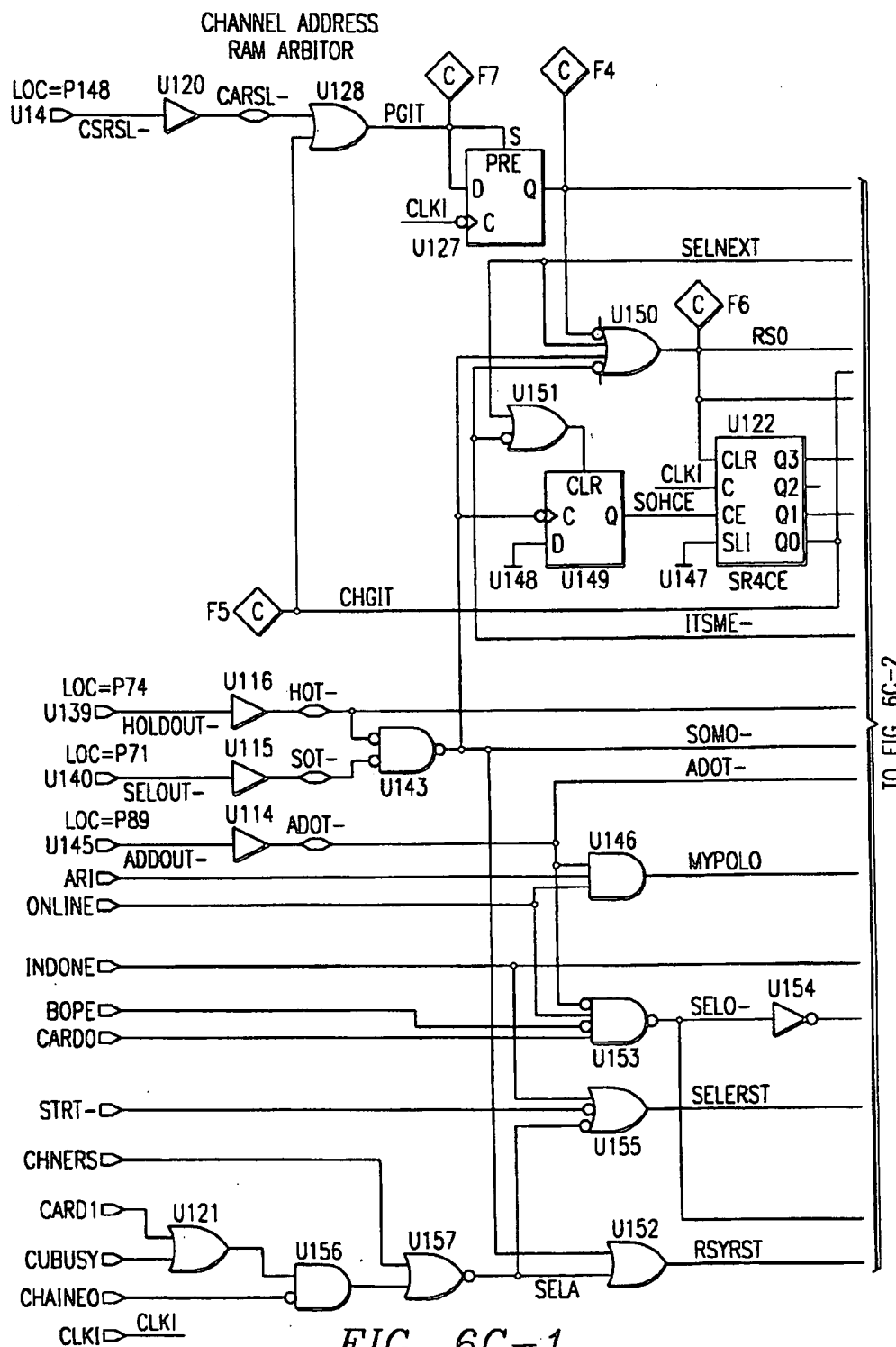
D8

D11

D14

D

SELECTION CONTROLLER



SELECTION CONTROLLER

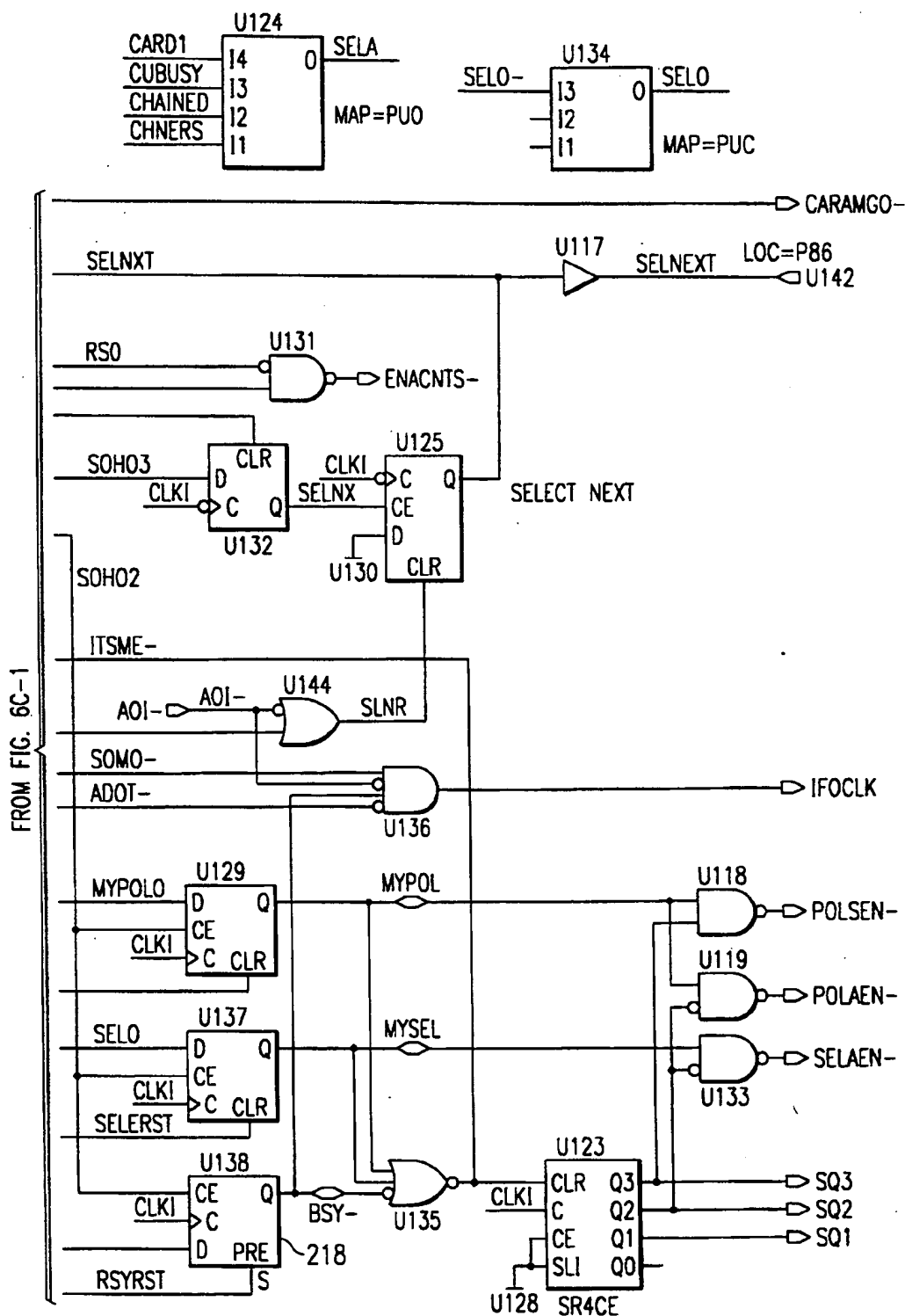
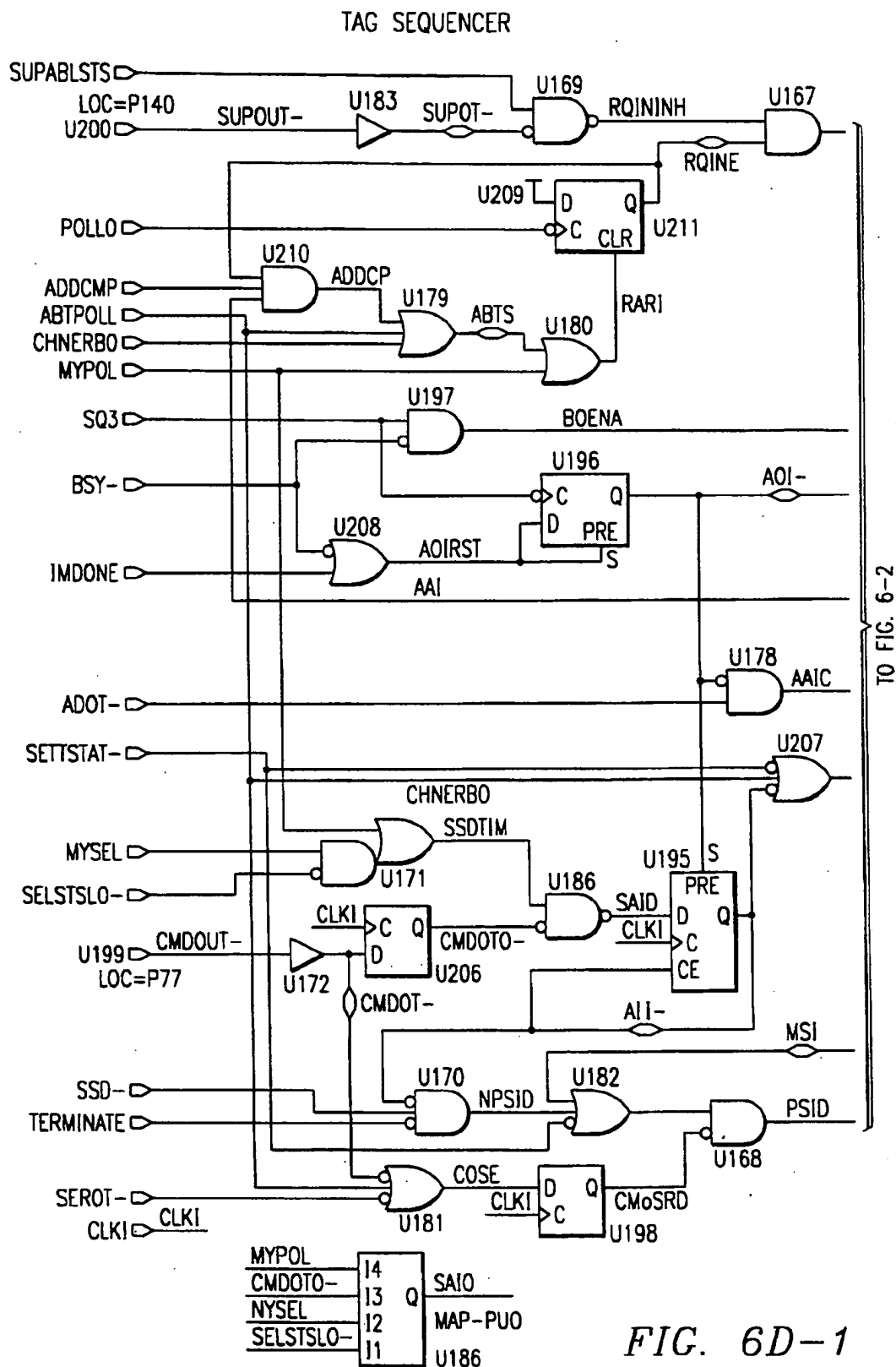
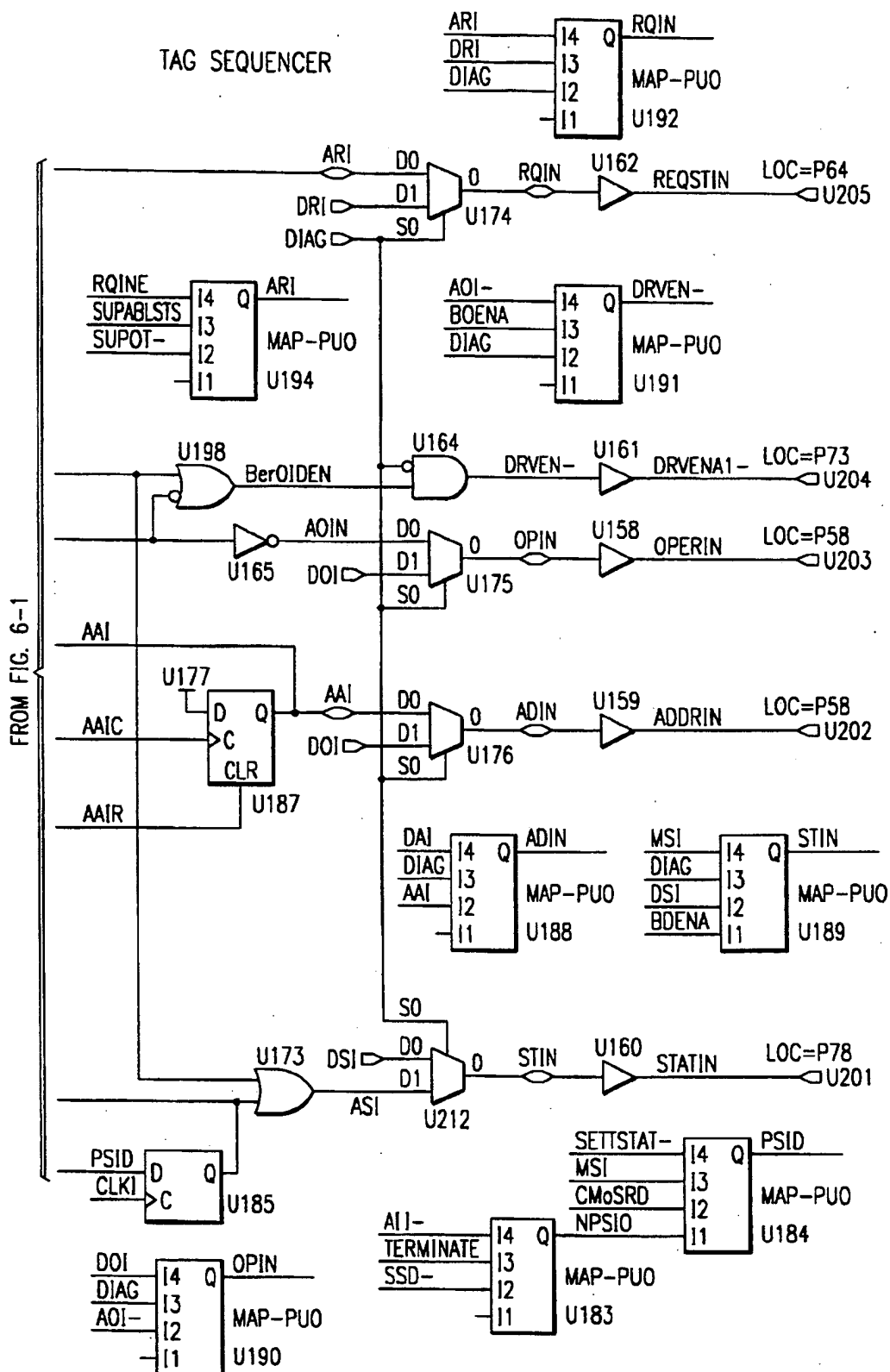


FIG. 6C-2





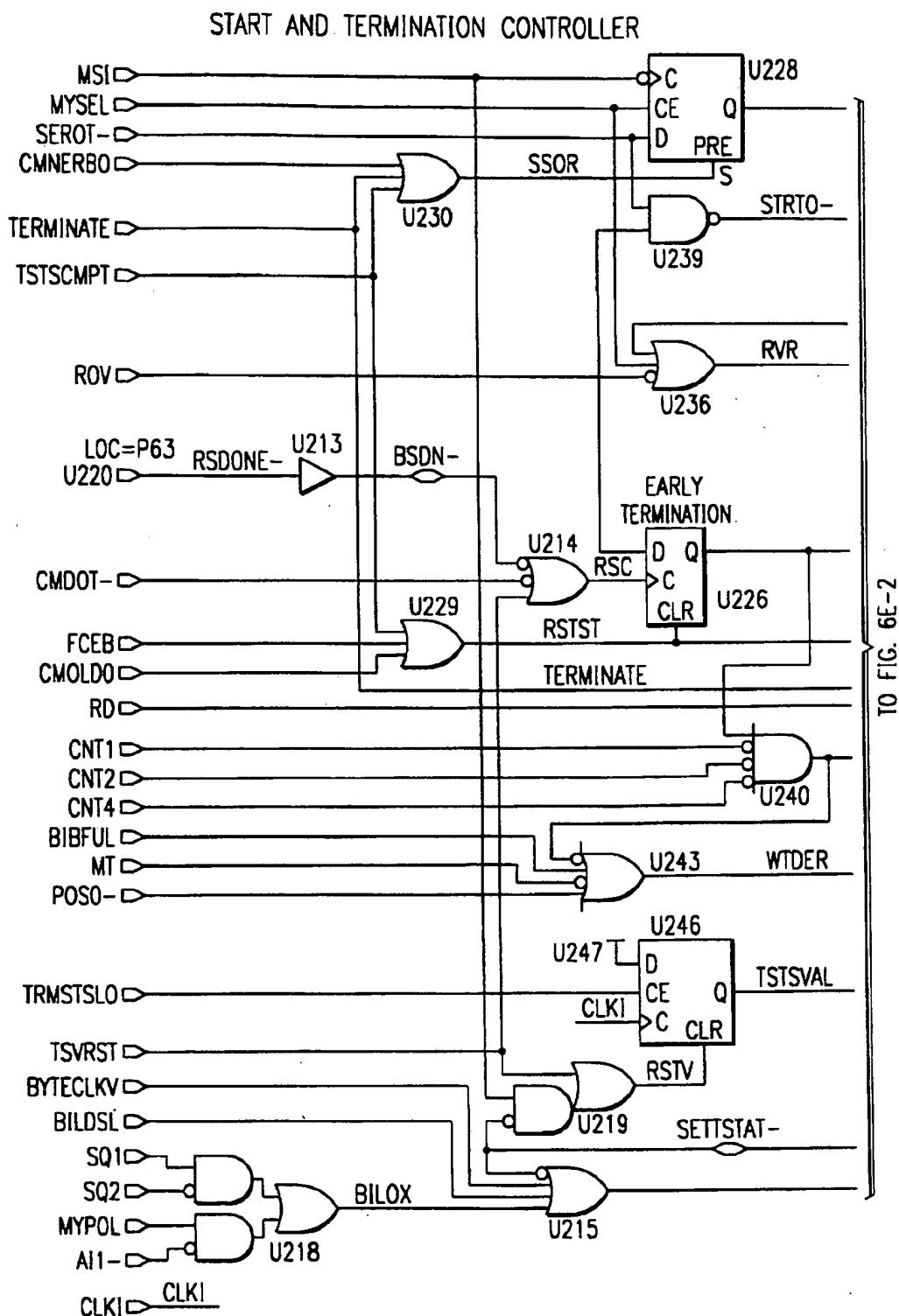
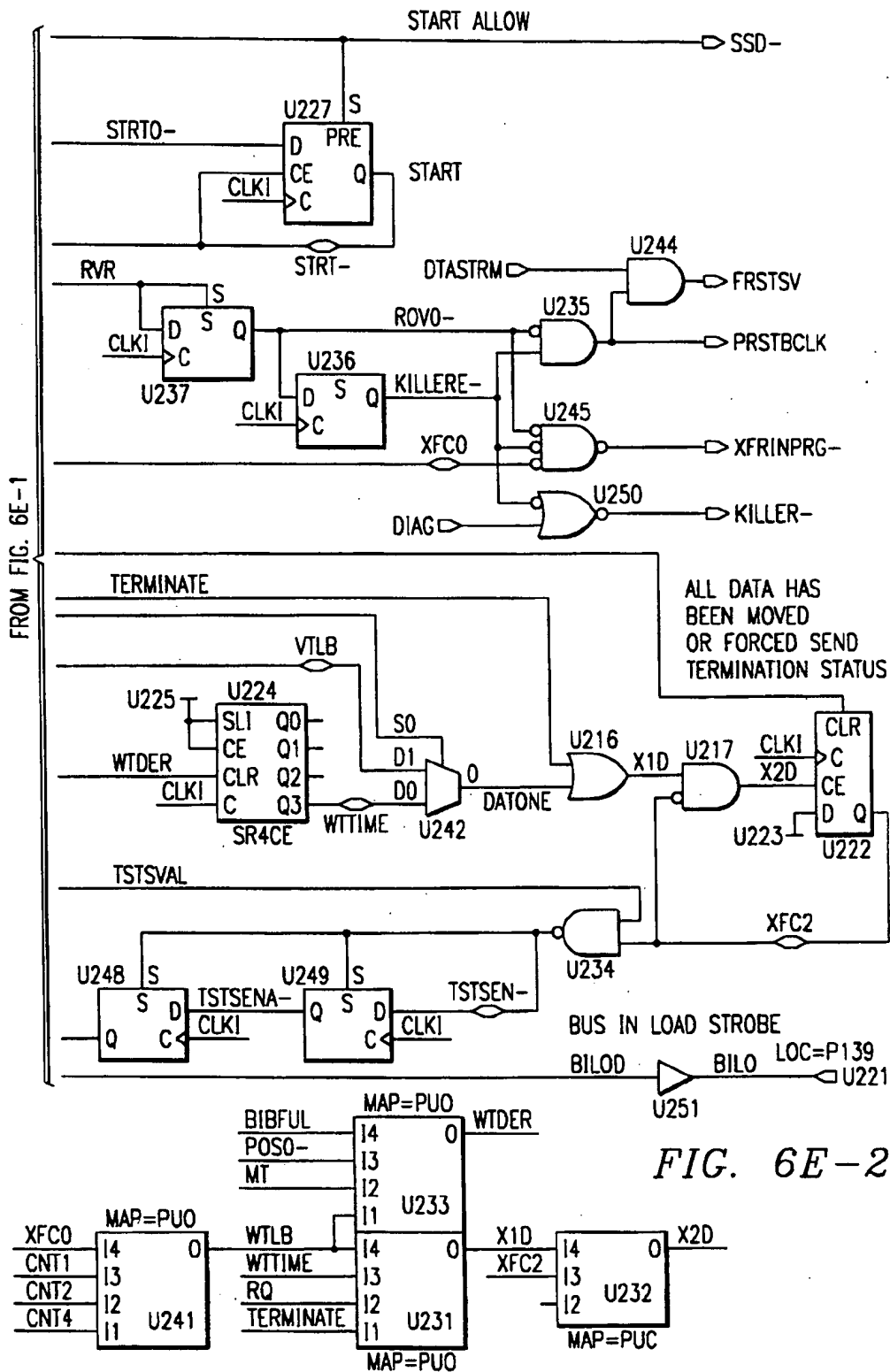
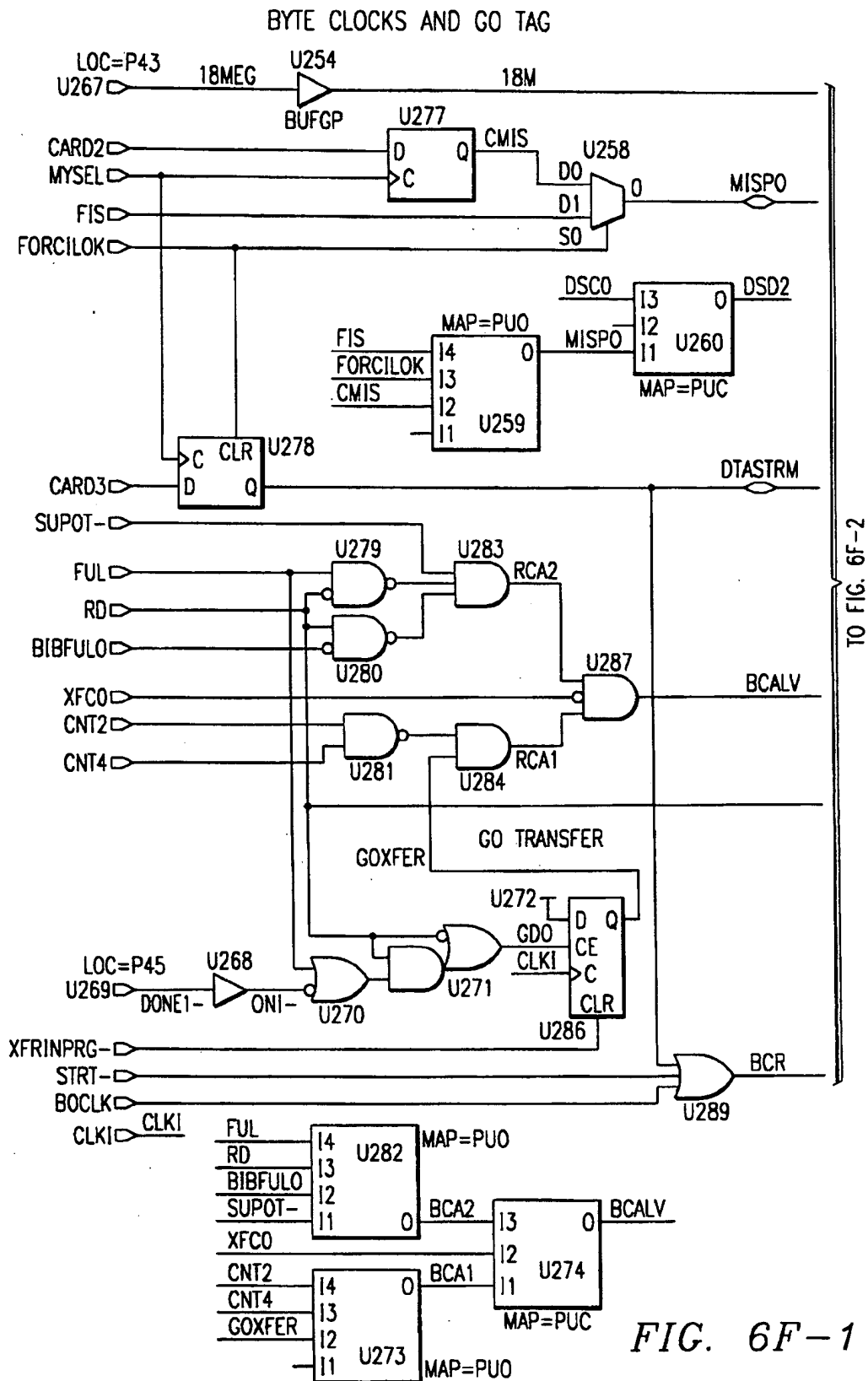


FIG. 6E-1

START AND TERMINATION CONTROLLER





BYTE CLOCKS AND GO TAG

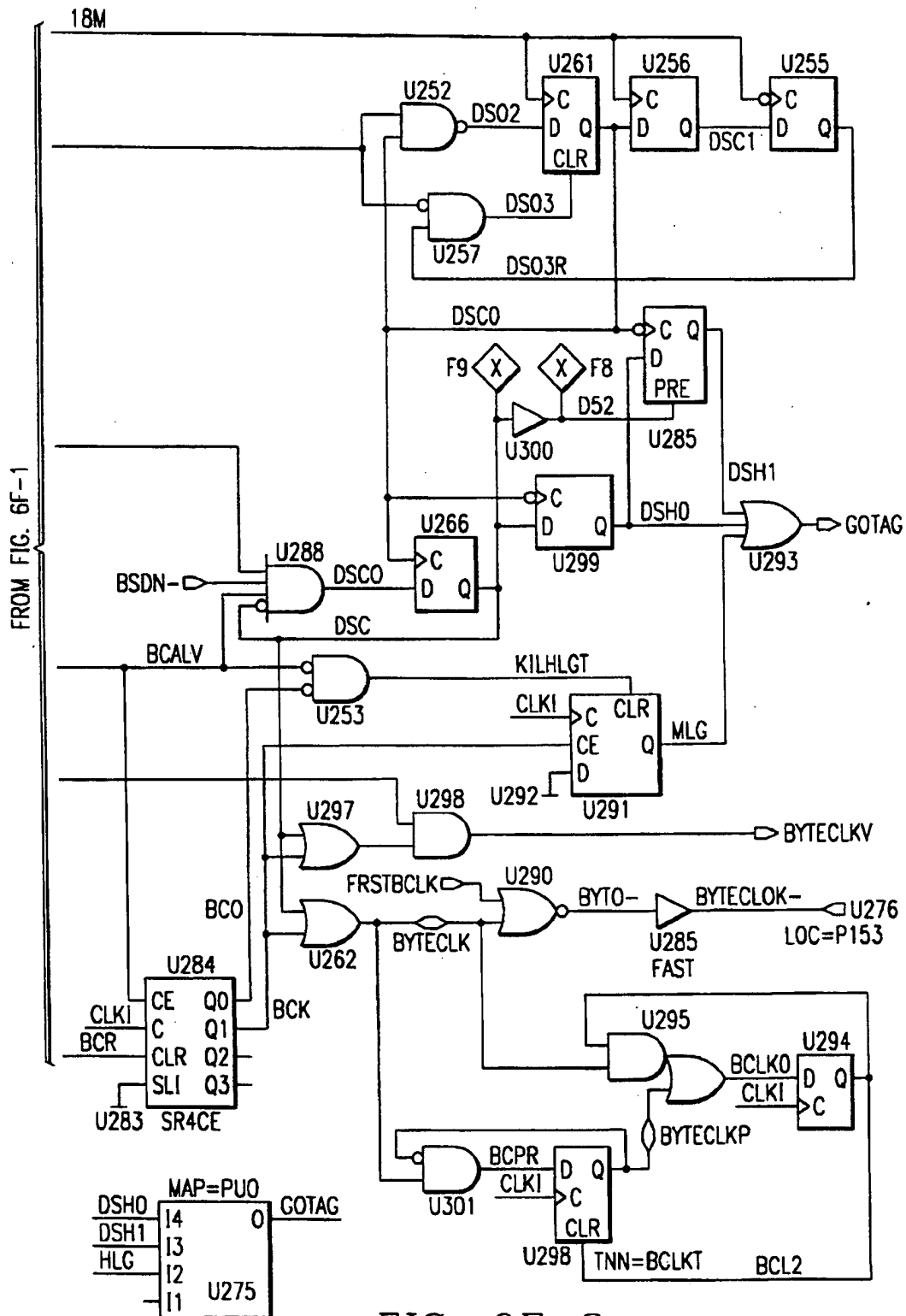
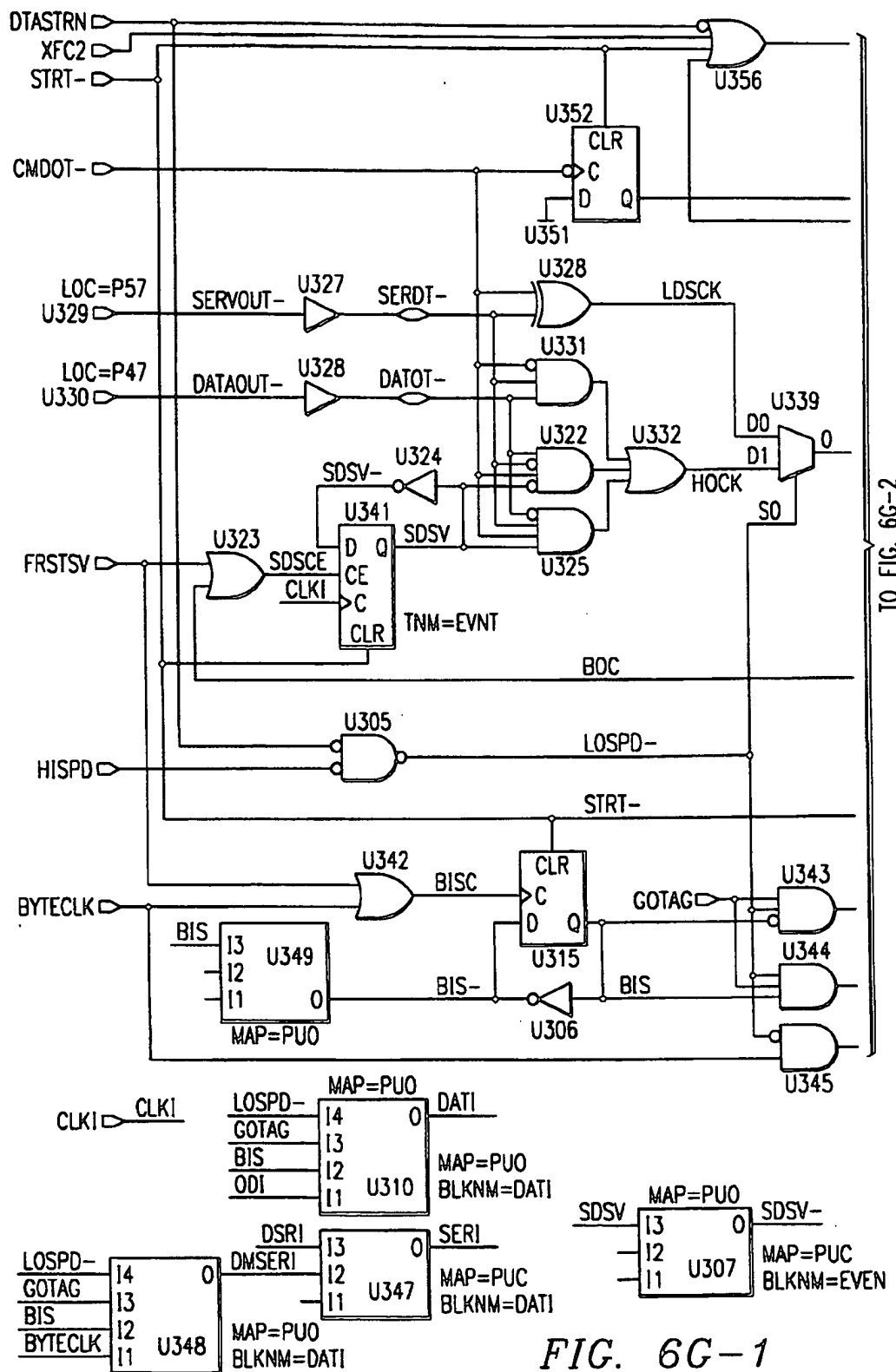


FIG. 6F-2

DATA TRANSFER CONTROLLER



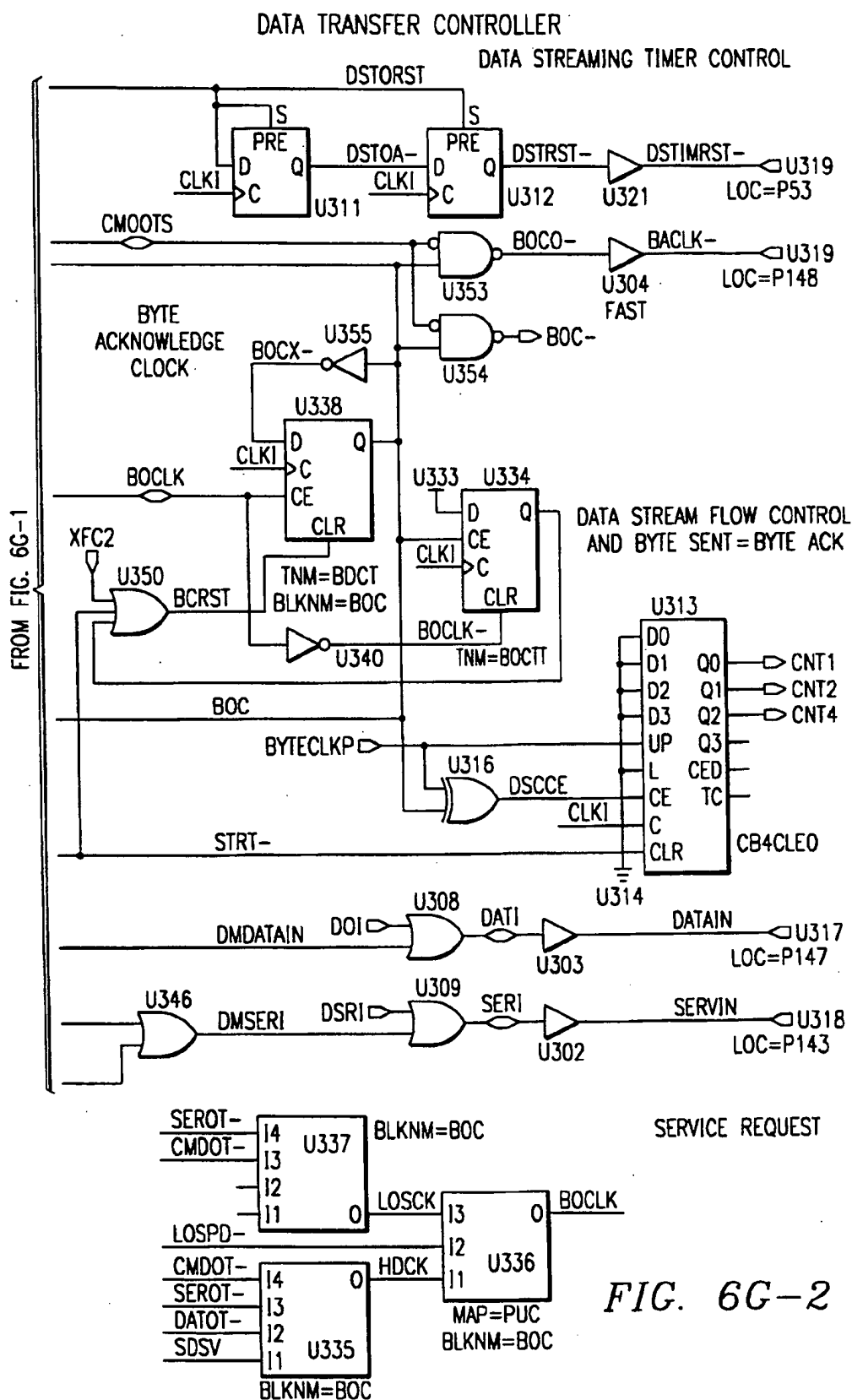
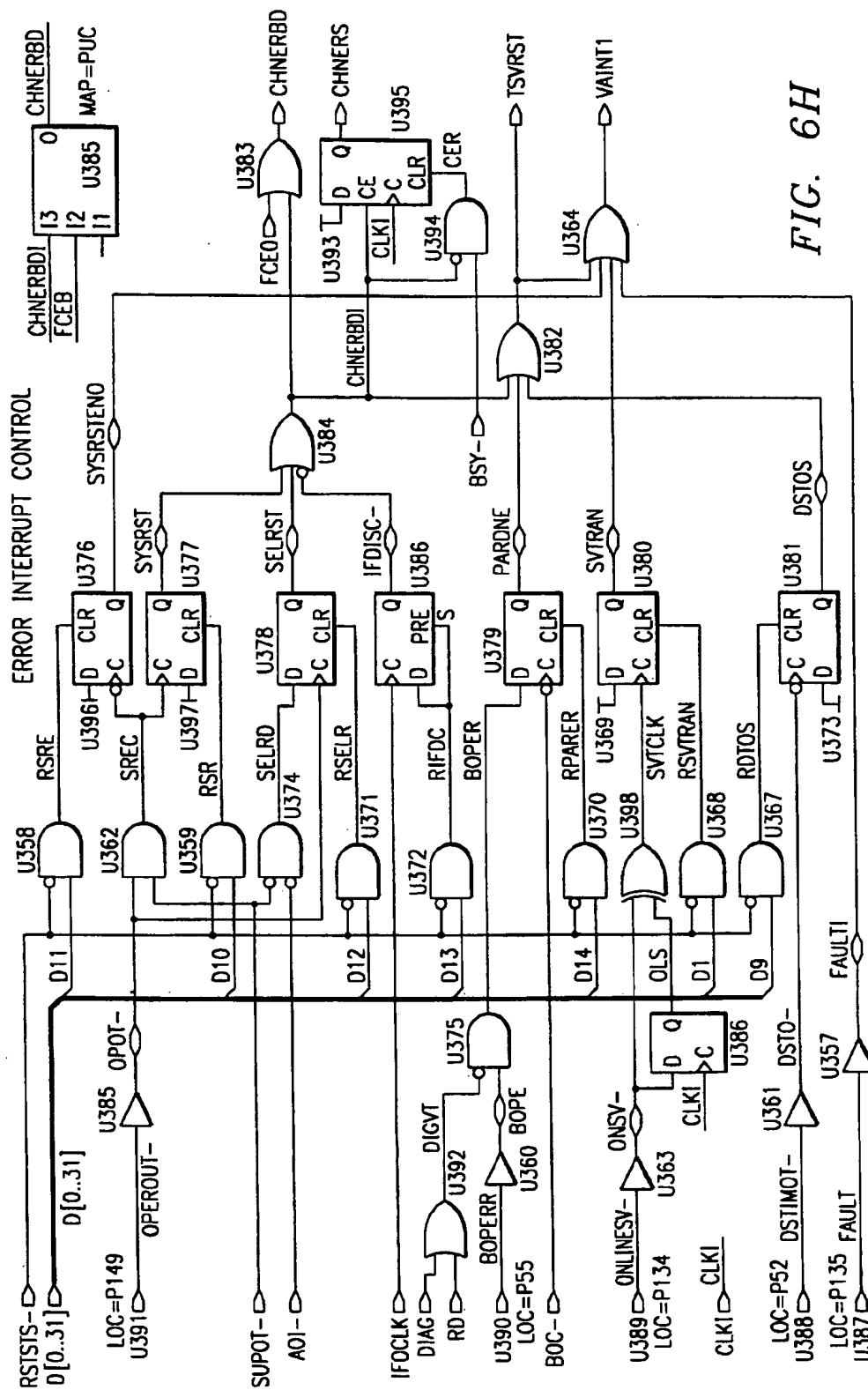


FIG. 6G-2



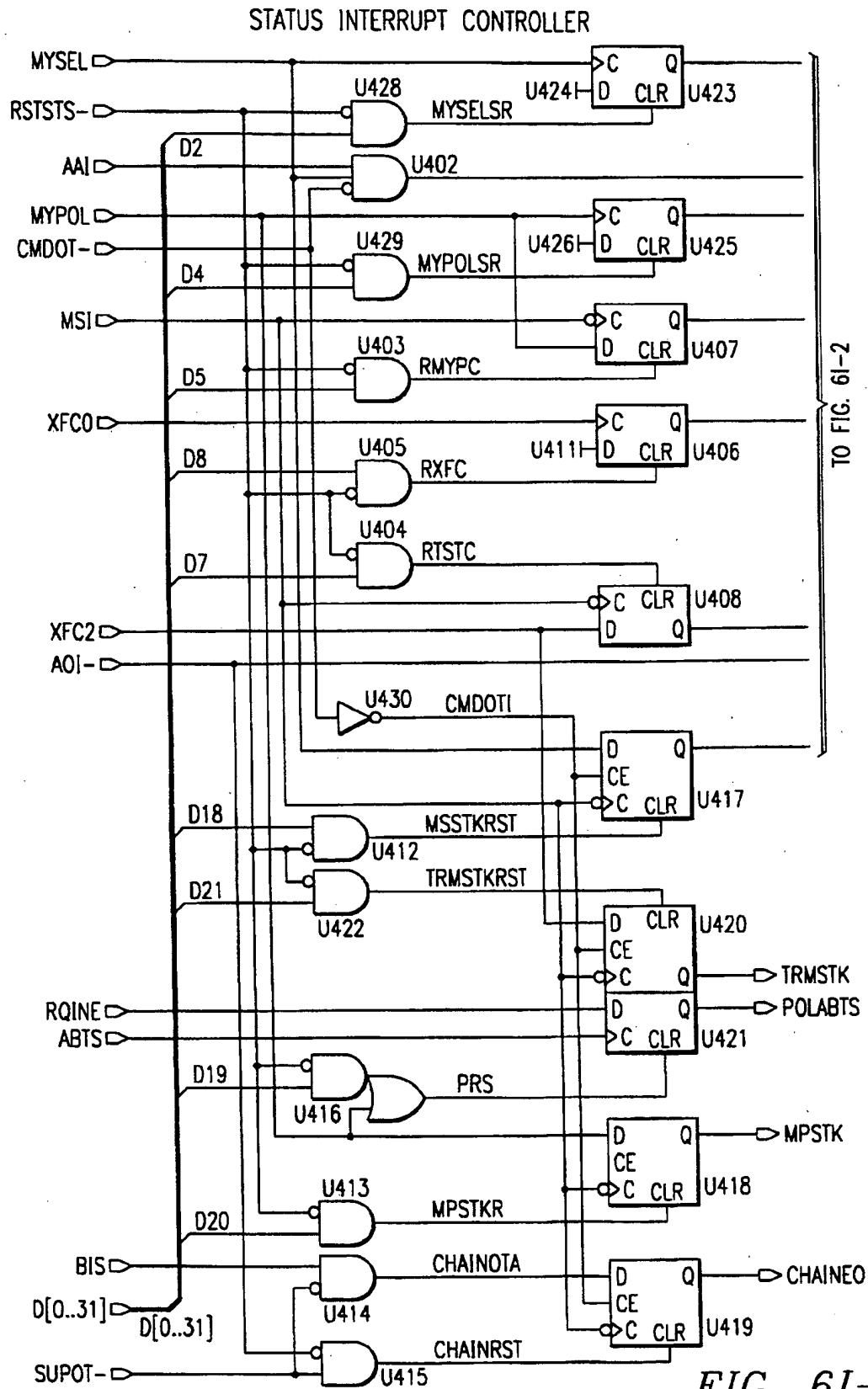
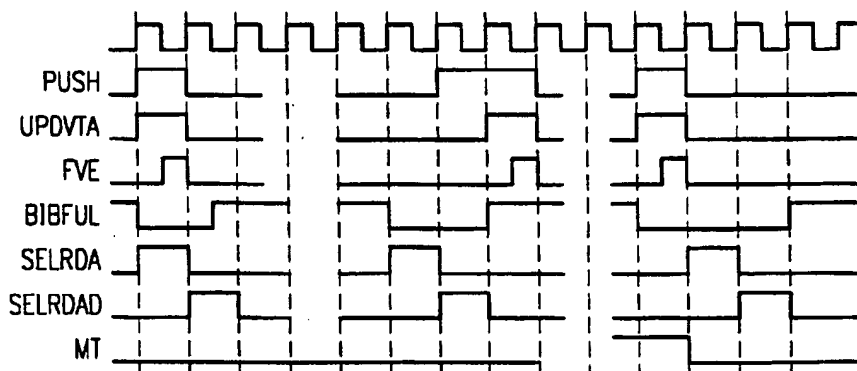
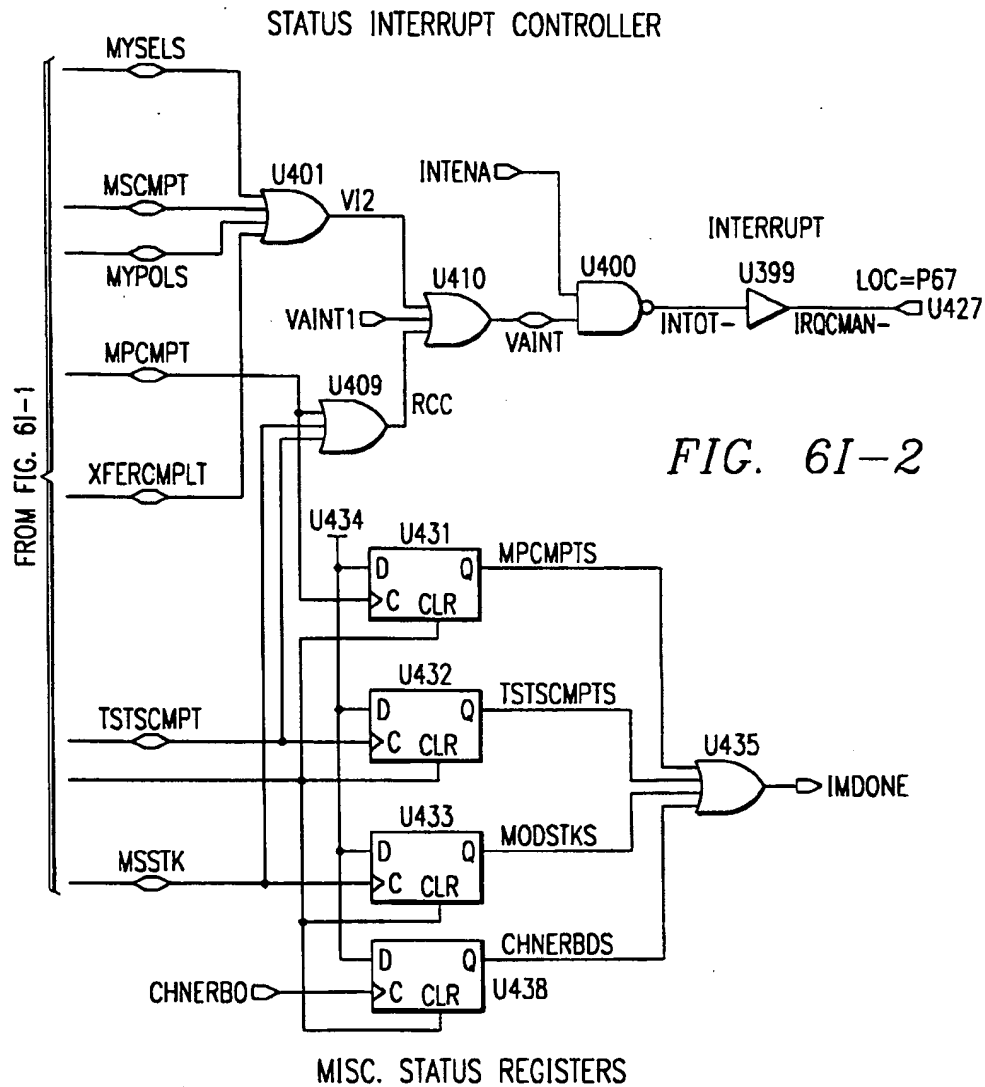


FIG. 61-1



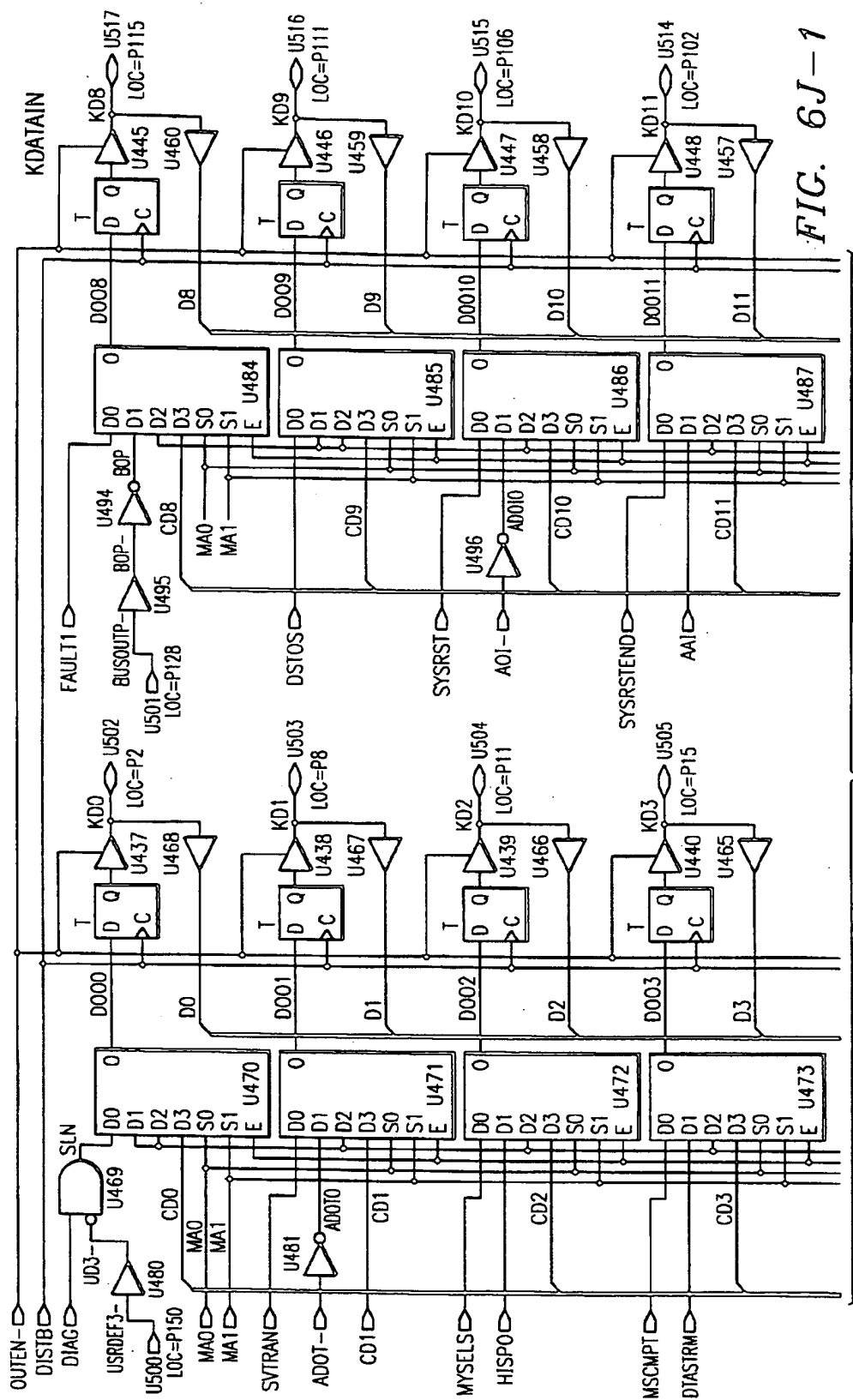
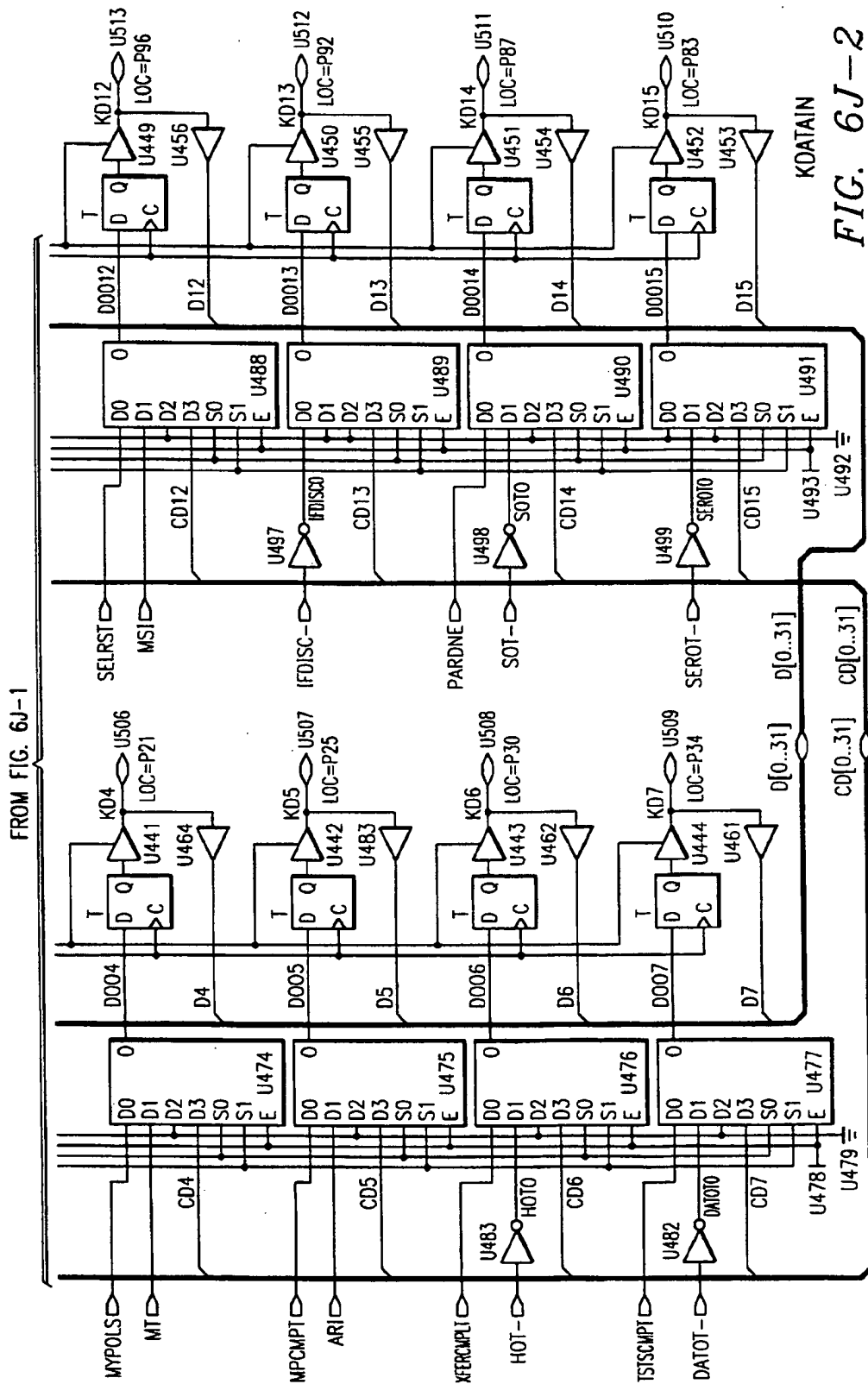
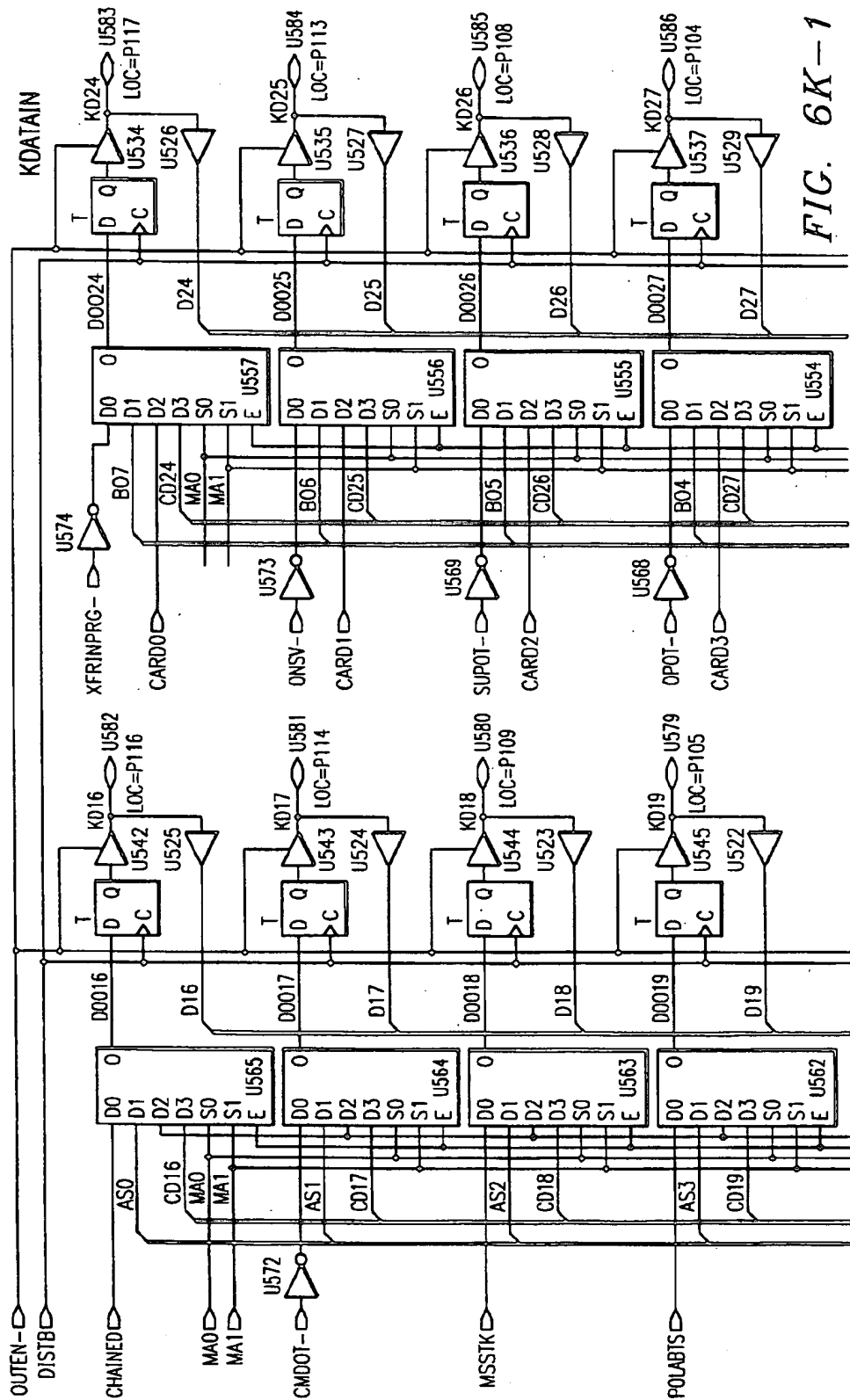


FIG. 6J-1

TO FIG. 6J-2





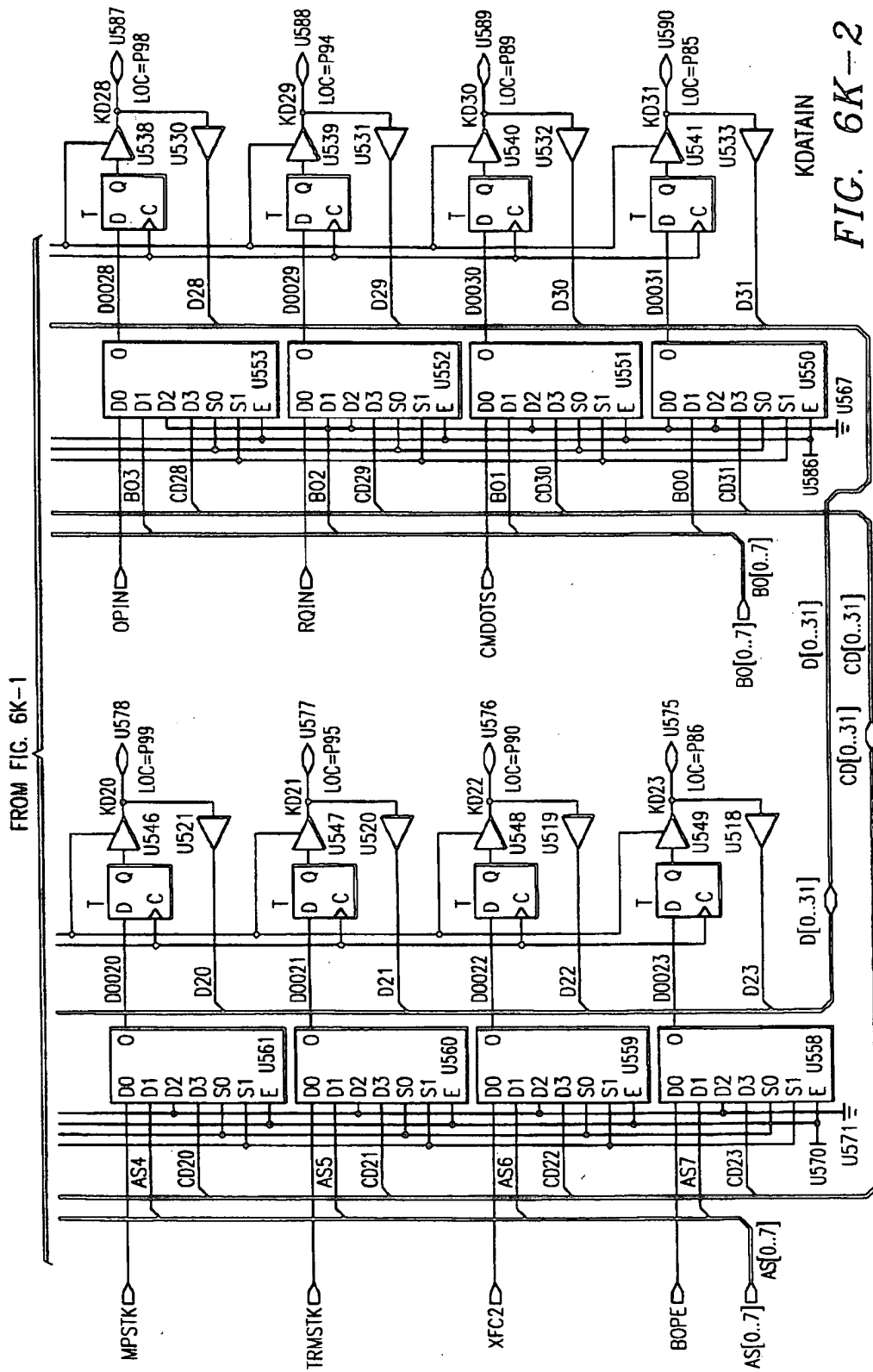


FIG. 6K-2

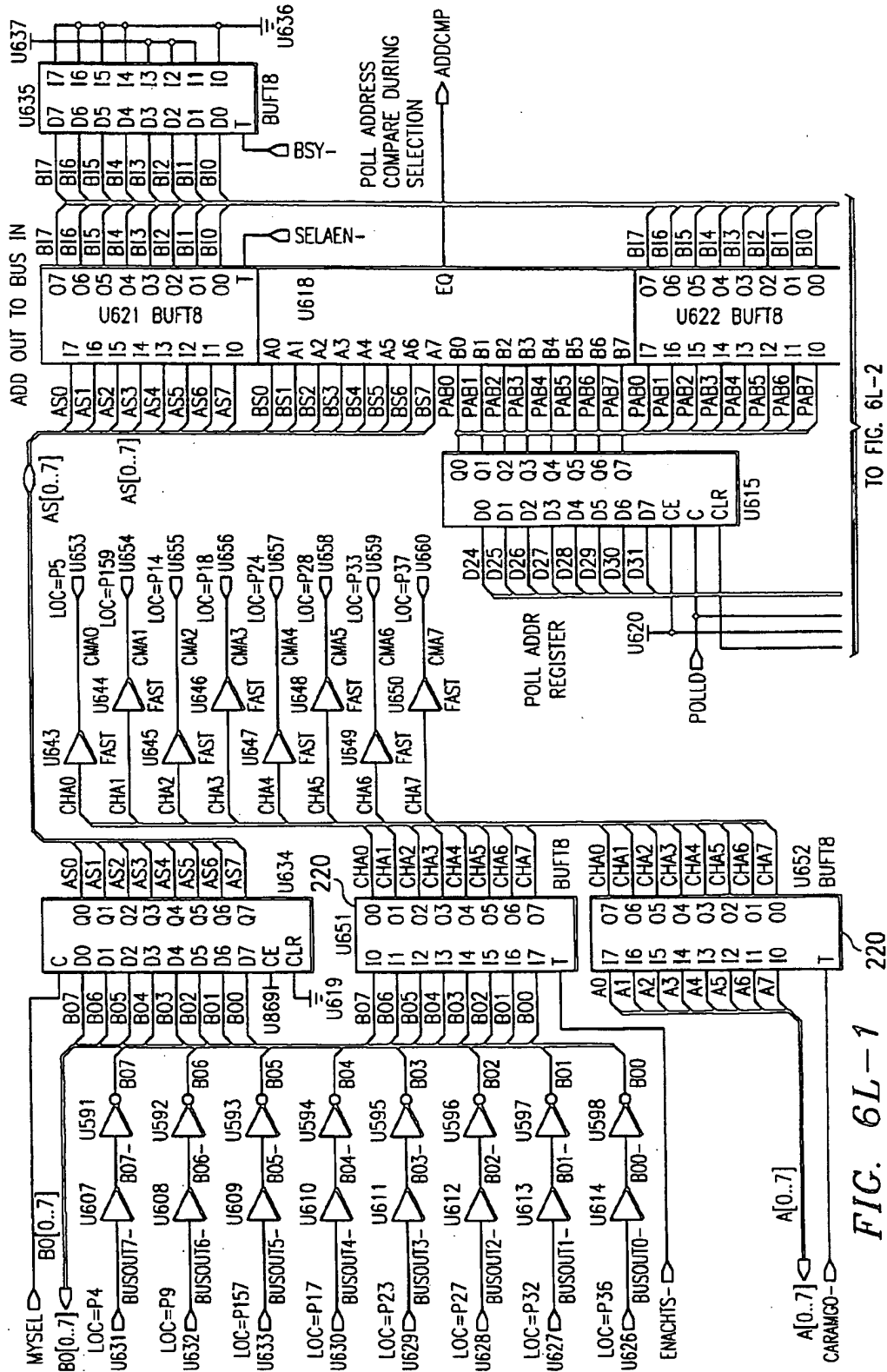
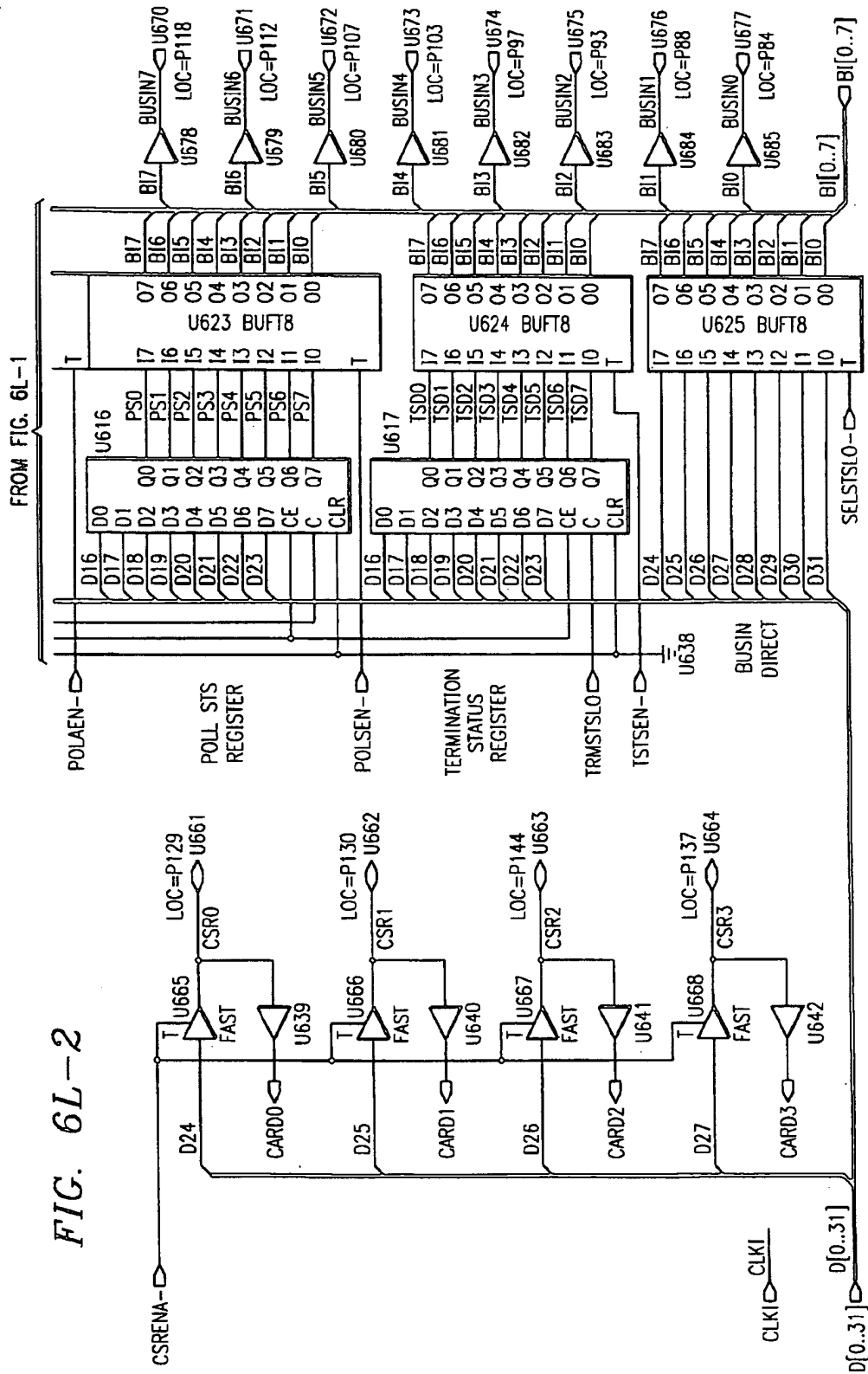


FIG. 6L-1

TO FIG. 6L-2



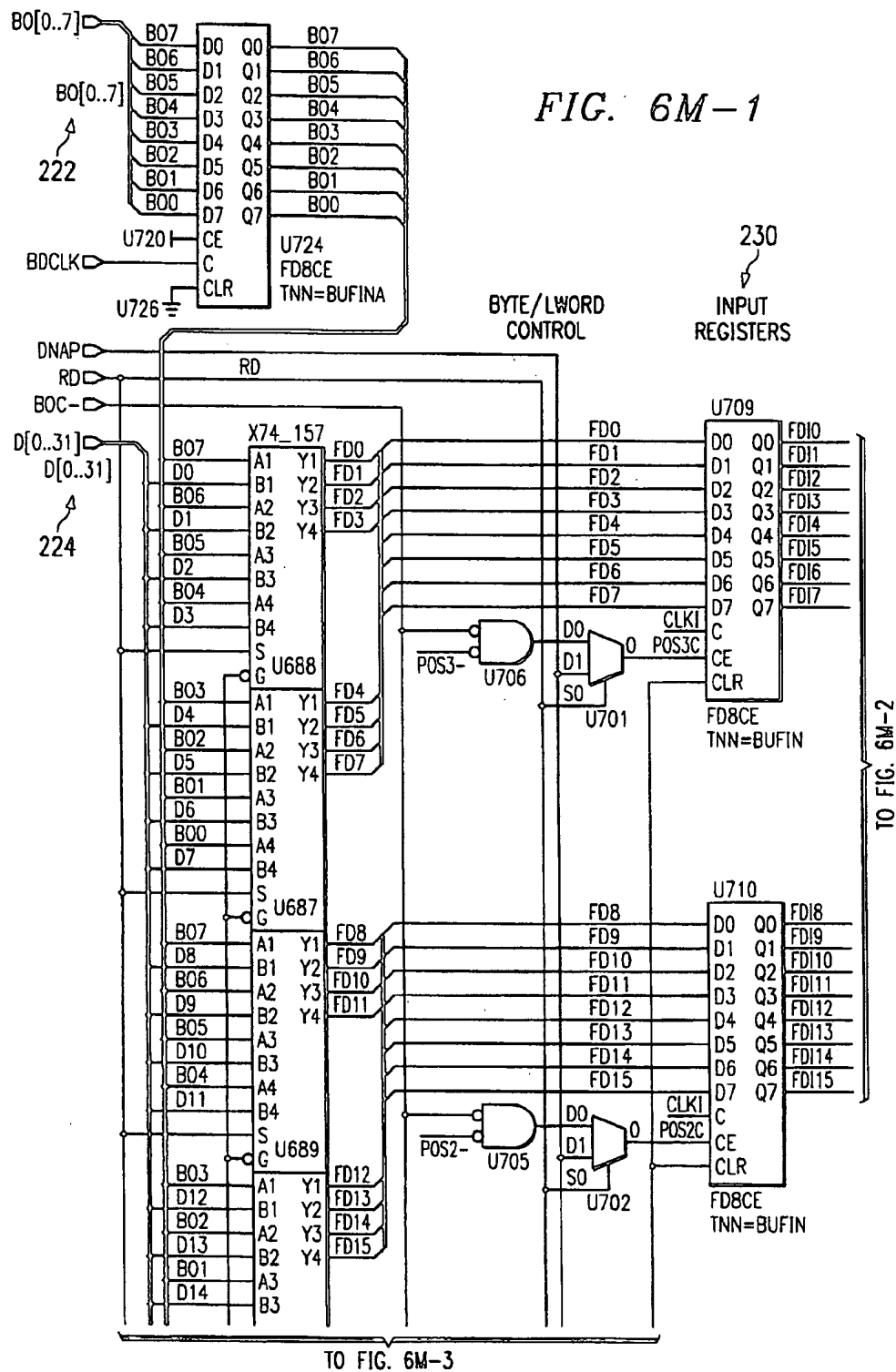
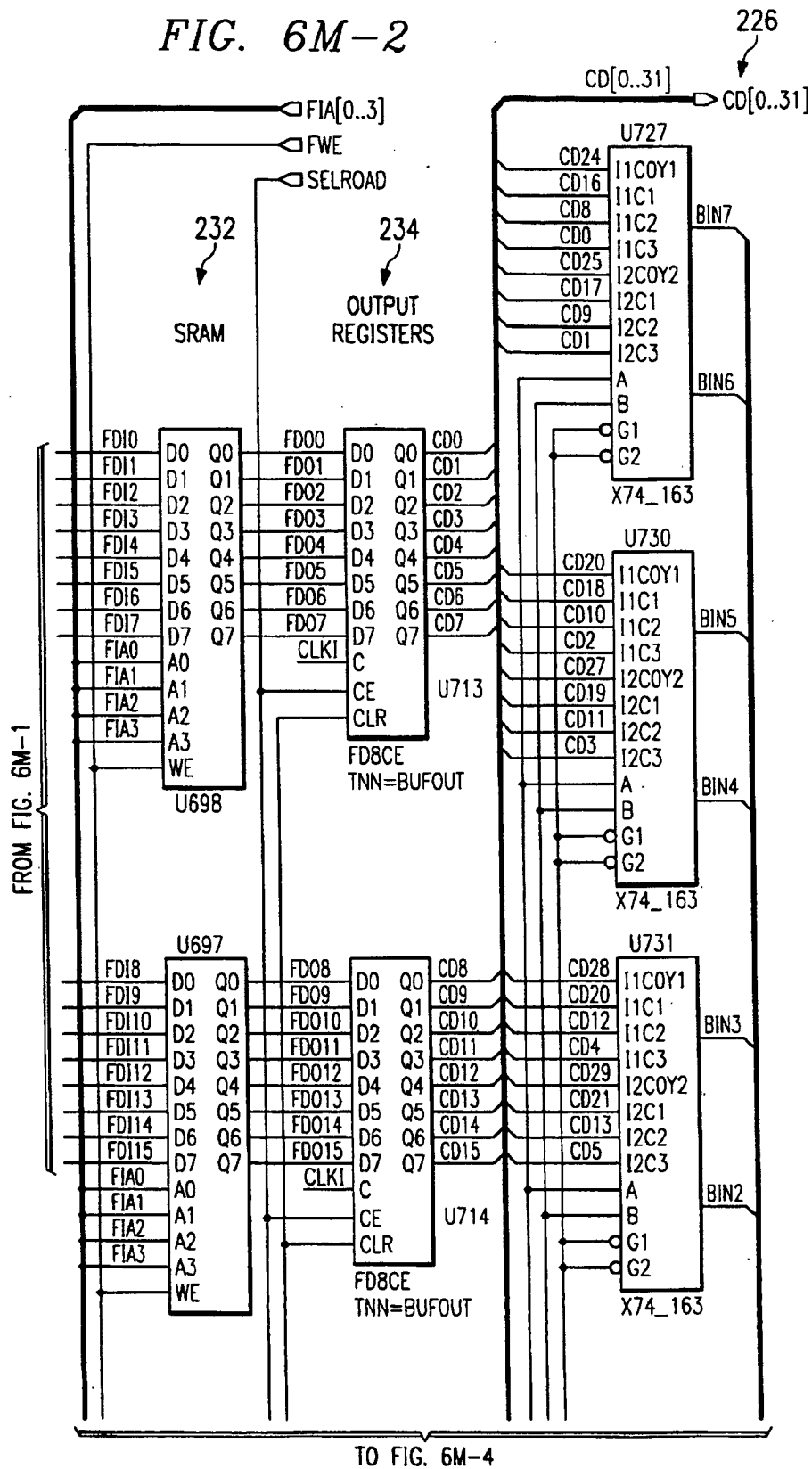
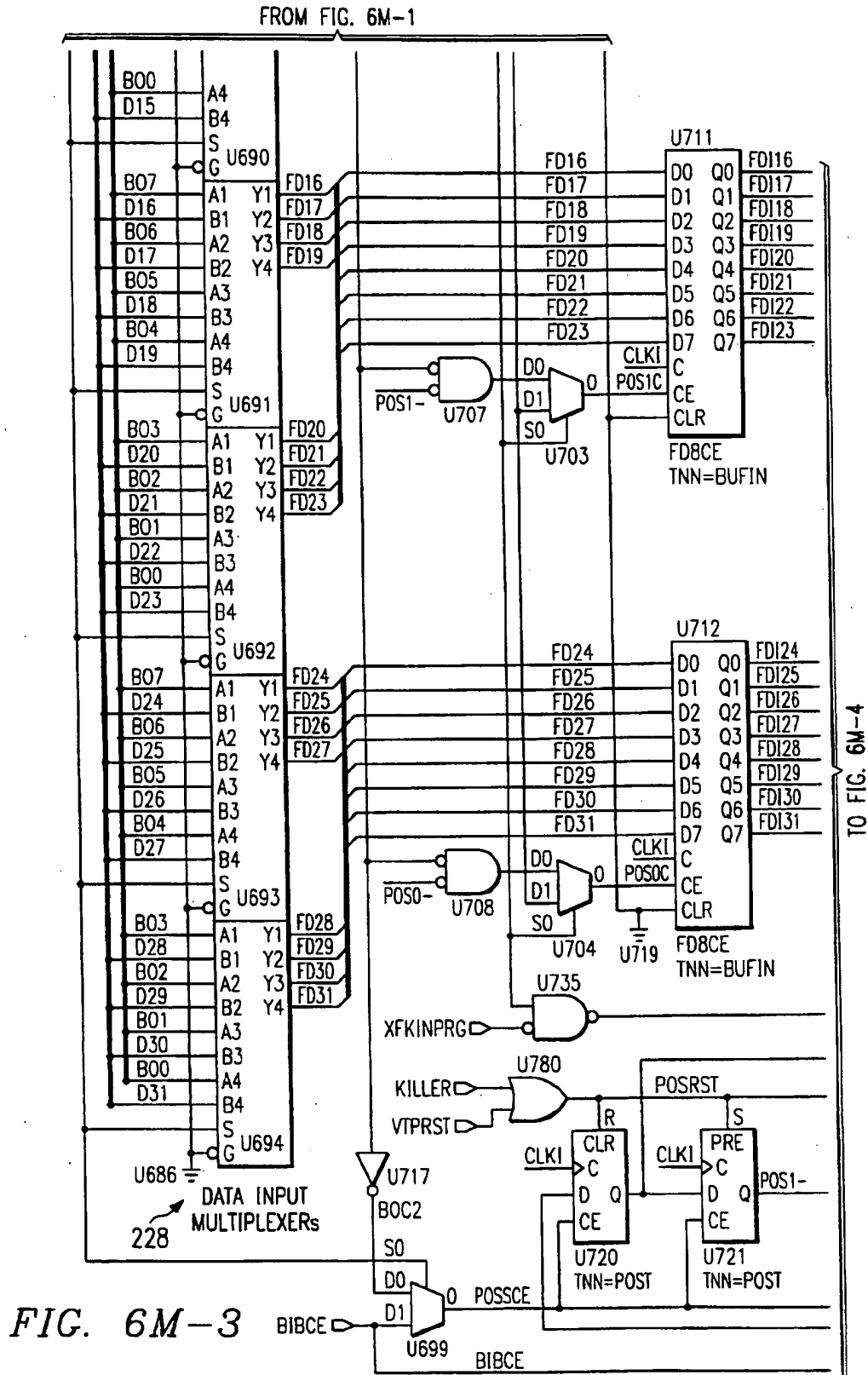


FIG. 6M-2





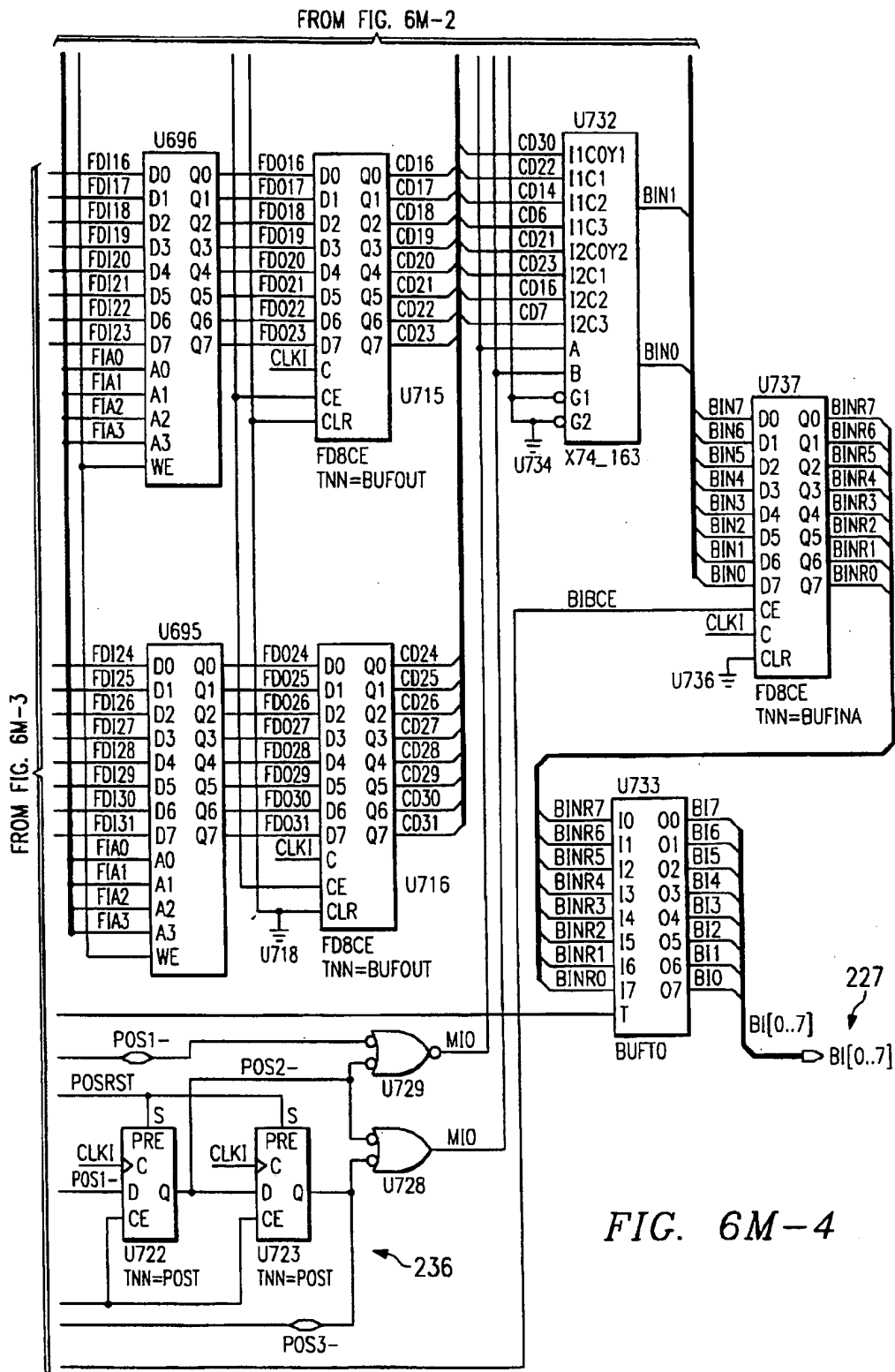


FIG. 6M-4

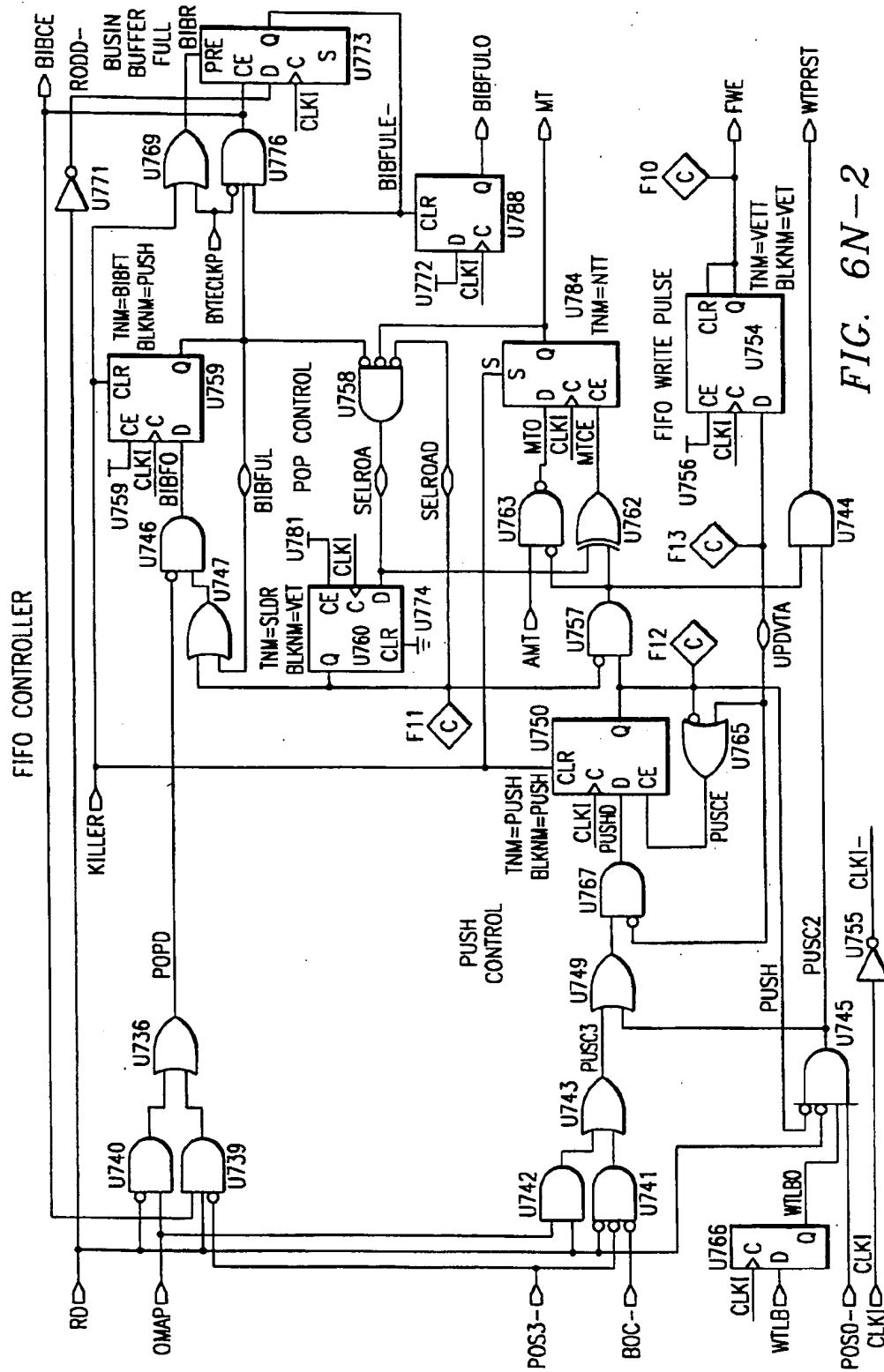
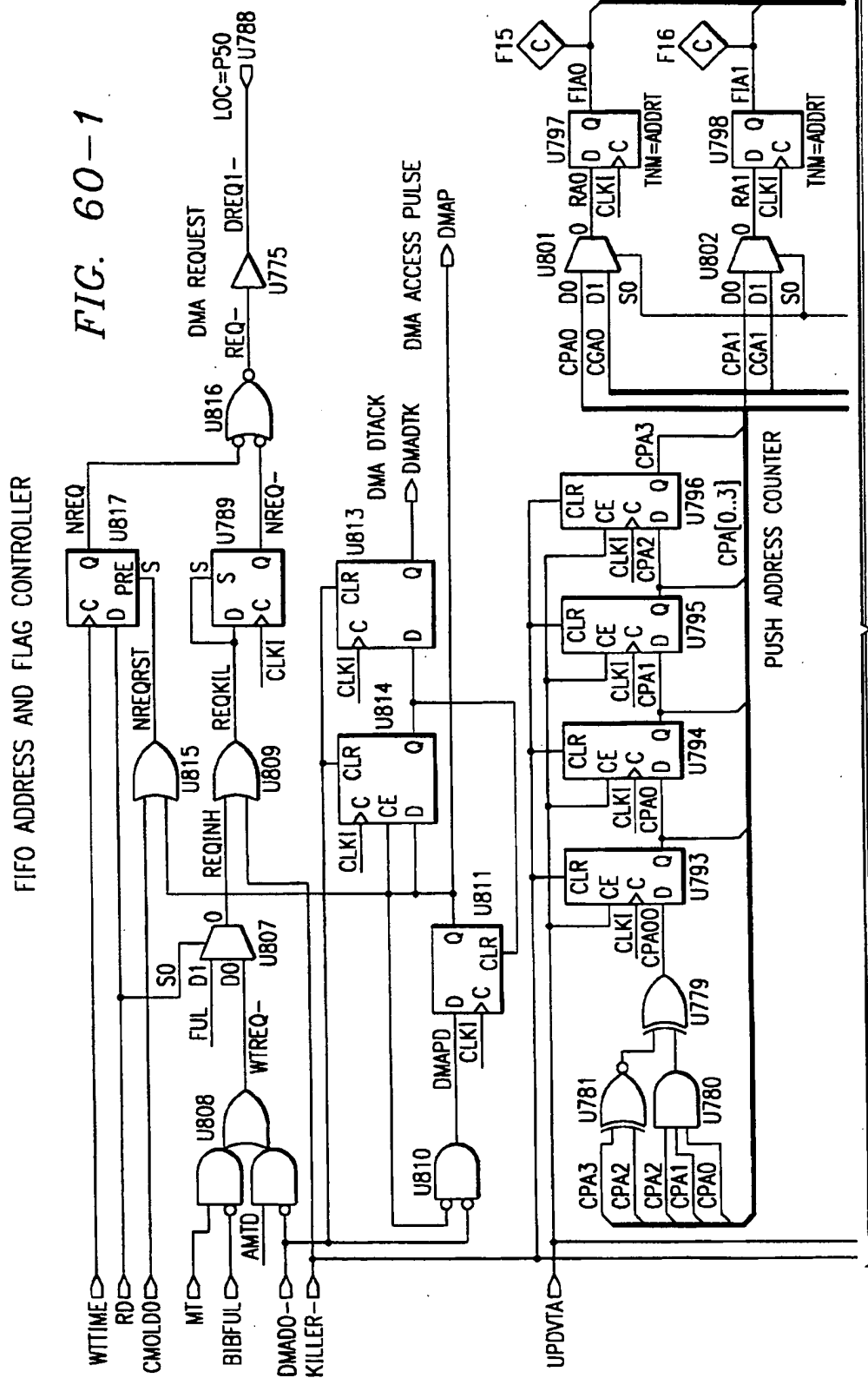


FIG. 6N-2

FIG. 60-1



TO FIG. 60-2

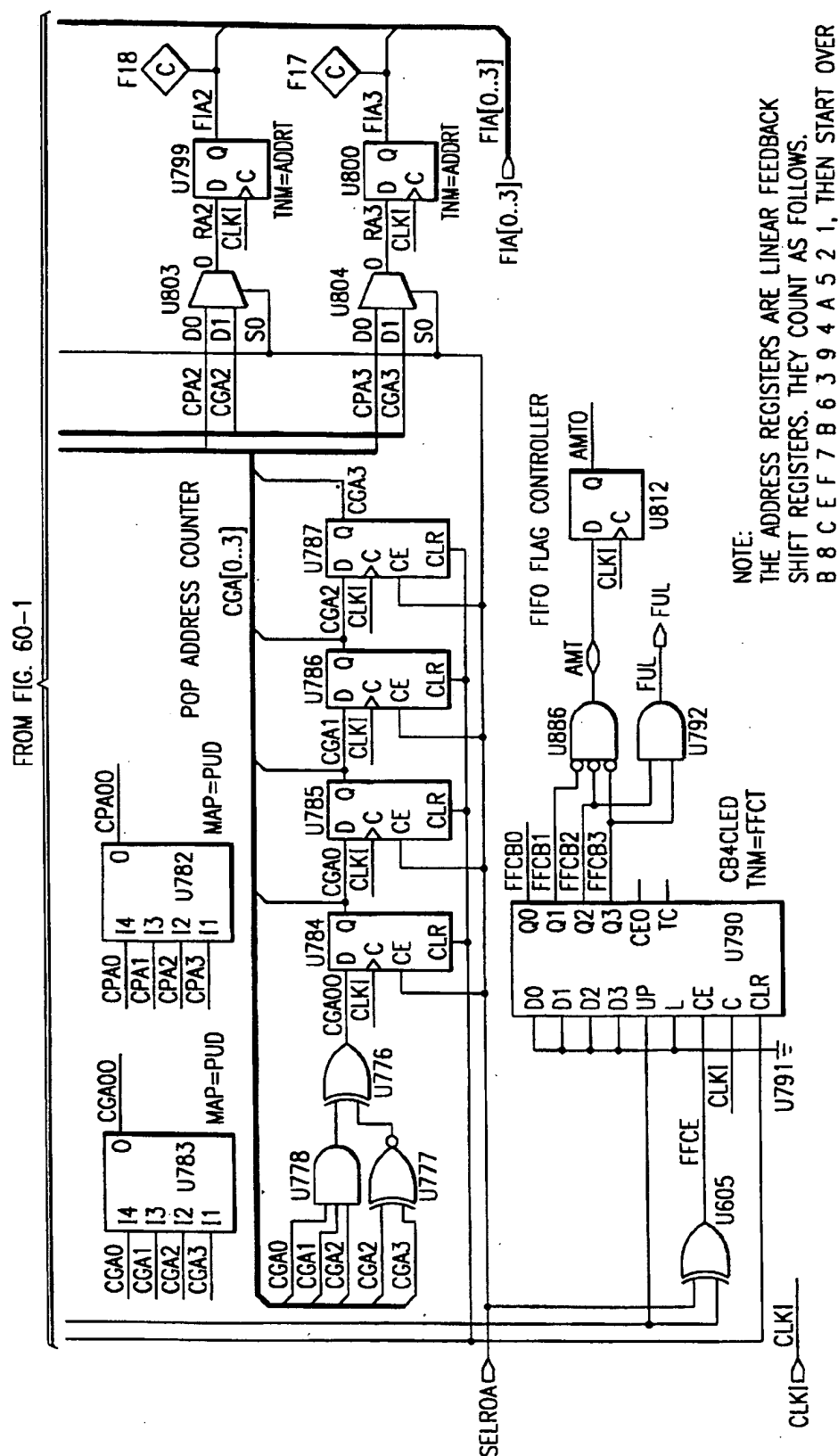


FIG. 60-2

ADAPTER FOR INTERFACING A SCSI BUS WITH AN IBM SYSTEM/360/370 I/O INTERFACE CHANNEL AND INFORMATION SYSTEM INCLUDING SAME

TECHNICAL FIELD OF THE INVENTION

This invention relates in general to the field of electronic systems, and more particularly to an adapter for interfacing a SCSI bus with an IBM System/360/370 I/O interface channel and an information system including such an adapter.

BACKGROUND OF THE INVENTION

Numerous public and private organizations use information systems that include two types of computer systems. The first type of computer system generally comprises IBM® mainframe and compatible systems supporting SYSTEM NETWORK ARCHITECTURE™ ("SNA") sometimes referred to as legacy systems. These legacy systems commonly support communication of information via an IBM System/360/370 I/O interface channel where the information is formatted according to various protocols including SNA. An IBM System/360/370 I/O interface channel can comprise a block-multiplexed or "bus & tag" channel providing approximately 4.5 megabytes per second of data transmission bandwidth or an ESCON fiber-optic channel providing approximately 17.5 megabytes per second of data transmission bandwidth.

The second type of computer system generally comprises computer workstations and personal computers such as UNIX®-based systems, DOS and WINDOWS™ systems, OS/2® systems and MACINTOSH® systems. These systems commonly support communication of information via a communication network where the information is formatted according to a TCP/IP protocol. Local area networks ("LAN's") and wide area networks ("WAN's") are often created by interconnecting these systems to form workgroup environments generally providing bandwidths in the range of 10–16 megabits per second. Computer workstations and personal computers also commonly support communication via a Small Computer Standard Interface ("SCSI") bus to provide standard connectivity for peripheral devices such as internal/external hard drives or tape drives.

It is important for an organization having an information system that includes both of these types of computer systems to be able to access and integrate information housed in legacy systems with information distributed throughout numerous workgroup environments of computer workstations and personal computers. Bidirectional movement of information and greater bandwidth are important considerations in providing this interconnectivity. It is desirable for information to travel in both directions between a legacy system and a computer workstation or personal computer and to do so at as large a bandwidth as possible. Currently, the bandwidth for such communication of information is limited by the bandwidth of the communication network to which the computer workstation or personal computer is connected.

SUMMARY OF THE INVENTION

In accordance with the present invention, disadvantages and problems associated with prior systems and methods for interfacing a computer system supporting communication via an IBM System/360/370 I/O interface channel with computer systems in multi-vendor local area networks have been substantially reduced or eliminated.

According to one embodiment of the present invention, an information system including a plurality of computer systems is provided. The information system includes a first computer system having an IBM System/360/370 I/O interface channel. The first computer system is operable to communicate SNA or channel protocol information via the IBM System/360/370 I/O interface channel. The information system includes a second computer system having a SCSI bus. The second computer system is operable to communicate SCSI protocol information via the SCSI bus. An adapter is coupled to the IBM System/360/370 I/O interface channel of the first computer system and the SCSI bus of the second computer system. The adapter is operable to interface the SCSI bus with the IBM System/360/370 I/O interface channel to allow bidirectional communication between the first computer system and the second computer system.

According to another embodiment of the present invention, an adapter for interfacing a SCSI bus with an IBM System/360/370 I/O interface channel is provided. The adapter includes a channel interface unit operable to couple to an IBM System/360/370 I/O interface channel. The channel interface unit is further operable to communicate SNA and IBM channel protocol information via the IBM System/360/370 I/O interface channel. The adapter also includes a SCSI interface unit operable to couple to a SCSI bus. The SCSI interface unit is further operable to communicate SCSI protocol information via the SCSI bus. A processor is coupled to the channel interface unit and to the SCSI interface unit. The processor is operable to control the channel interface unit and the SCSI interface unit to allow bidirectional communication between the SCSI bus and the IBM System/360/370 I/O interface channel.

Technical advantages of the present invention include providing an adapter for interfacing a SCSI bus with an IBM System/360/370 I/O interface channel to allow bidirectional communication taking advantage of the bandwidth of the IBM System/360/370 I/O interface channel. Because of the wide availability of SCSI device ports on computer workstations and personal computers, an adapter constructed according to the teachings of the present invention benefits numerous information systems currently used by organizations that include both SNA and TCP/IP environments.

BRIEF DESCRIPTION OF THE DRAWINGS

A more complete understanding of the present invention and advantages thereof may be acquired by reference to the following description taken in conjunction with the accompanying drawings in which like reference numbers indicate like features and wherein:

FIG. 1 illustrates one embodiment of an information system including an adapter for interfacing a SCSI bus with an IBM System/360/370 I/O interface channel constructed according to the teachings of the present invention;

FIG. 2 is a block diagram of one embodiment of an IBM® mainframe connected to a SCSI host via an adapter for interfacing a SCSI bus with an IBM System/360/370 I/O interface channel constructed according to the teachings of the present invention;

FIG. 3 is a block diagram of one embodiment of the adapter for interfacing a SCSI bus with an IBM System/360/370 I/O interface channel of FIG. 2;

FIGS. 4A and 4B are block diagrams of one embodiment of software and firmware operating in the SCSI host and the adapter for interfacing a SCSI bus with an IBM System/360/370 I/O interface channel of FIGS. 2 and 3;

FIGS. 5A, 5B-1, 5B-2, 5B-3, 5B-4 and 5C-1, 5C-2, 5D, 5E-1, 5E-2, 5F-1, 5F-2, 5G, 5H-1, 5H-2, 5H-3, 5I, 5J-1, 5J-2, 5J-3 and 5K are schematics showing one embodiment of the components and interconnections for the adapter for interfacing a SCSI bus with an IBM System/360/370 I/O interface channel of FIGS. 2 through 4; and

FIGS. 6A-1, 6A-2, 6B, 6C-1, 6C-2, 6D-1, 6D-2, 6E-1, 6E-2, 6F-1, 6F-2, 6G-1, 6G-2, 6H, 6I-1, 6I-2, 6J-1, 6J-2, 6K-1, 6K-2, 6L-1, 6L-2, 6M-1, 6M-2, 6M-3, 6M-4, 6N-1, 6N-2, and 6O-1, 6O-2 are schematics showing one embodiment of the logic components and interconnections for a channel controller of the adapter for interfacing a SCSI bus with an IBM System/360/370 I/O interface channel of FIGS. 2 through 4.

DETAILED DESCRIPTION OF THE INVENTION

Information System having SNA and TCP/IP environments

FIG. 1 illustrates one embodiment of an information system, indicated generally at 10. Information system includes a SYSTEM NETWORK ARCHITECTURE™ ("SNA") environment, indicated generally at 12, and a TCP/IP environment, indicated generally at 14. SNA environment 12 includes a mainframe 16. In the embodiment of FIG. 1, mainframe 16 comprises an IBM® mainframe using an MVS, VSE or VM operating system. SNA environment 12 can include additional compatible computer systems supporting SNA and an IBM System/360/370 I/O interface channel. Mainframe 16 has a number of IBM System/360/370 I/O interface channels 18 accessible through IBM System/360/370 I/O interface channel ports in mainframe 18.

A front-end processor 20 is coupled to a channel 18, as shown. Front-end processor 20 is coupled to a number of terminal networks including an SDLC network 22, an X.25 network 24 and a Token Ring network 26. Front-end processor 20 can be coupled to other types terminal networks as well. A first hardware gateway device 28 is coupled to SDLC network 22. A computer system 30 operating as a gateway device and a second hardware gateway device 32 are coupled to Token Ring network 26. A third hardware gateway device 34 is coupled to a second channel 18 of mainframe 16, as shown. In the embodiment of FIG. 1, first gateway device 28, second gateway device 32 and third gateway device 34 comprise OC SERVER I standalone hardware gateways available from OPENCONNECT® SYSTEMS located in Dallas, Tex. In the embodiment of FIG. 1, workstation 30 comprises a UNIX®-based workstation running OC SERVER II software gateway available from OPENCONNECT® SYSTEMS located in Dallas, Tex.

An adapter 36 is coupled to a third channel 18 of mainframe 16. Adapter 36 is also coupled to a Small Computer Standard Interface ("SCSI") bus 38 of computer system 40. SCSI bus 38 is accessible through a SCSI device port of computer system 40. Adapter 36 operates to interface SCSI bus 38 with channel 18 according to the teachings of the present invention.

TCP/IP environment 14 includes a TCP/IP network 42. TCP/IP network 42 is coupled to communication units of first gateway device 28, computer system 30, second gateway device 32, third gateway device 34 and computer system 40. TCP/IP environment 14 includes a first workgroup environment having computer systems 44 and printer 46, each of which has a communication unit coupled to TCP/IP network 42. TCP/IP environment 14 includes a second workgroup environment having computer systems

48 and terminal server 50 supplying a wide area network ("WAN"), each of which has a communication unit coupled to TCP/IP network 42. In the embodiment of FIG. 1, computer systems 44 and computer systems 48 may comprise, for example, multi-vendor computer workstations and personal computers.

In operation, mainframe 16 can communicate with computer systems 44, printer 46, computer systems 48 and terminal server 50 through first gateway device 28, computer system 30, second gateway device 32, and third gateway device 34. However, the bandwidth of this communication is limited by the bandwidth of SDLC network 22, X.25 network 24 Token Ring network 26 and TCP/IP network 42.

Mainframe 16 also can communicate with computer system 40 through adapter 36. Computer system 40 can communicate with computer systems 44, printer 46, computer systems 48 and terminal server 50 through TCP/IP network 42. Adapter 36 is constructed according to the teachings of the present invention and is operable to interface SCSI bus 38 with IBM System/360/370 I/O interface channel 18 enabling high bandwidth bidirectional communication between computer system 40 and mainframe 16.

Adapter interfacing SCSI bus with IBM System/360/370 I/O interface channel

FIG. 2 is a block diagram of one embodiment of an IBM® mainframe 16 connected to a SCSI host 40 via an adapter 36 for interfacing SCSI bus 38 with IBM System/360/370 I/O interface channel 18 constructed according to the teachings of the present invention.

In the embodiment of FIG. 2, SCSI host 40 comprises a UNIX®-based computer workstation. Mainframe 16 has an IBM System/360/370 I/O interface channel port 51 through which IBM System/360/370 I/O interface channel 18 is accessible. Adapter 36 has an IBM System/360/370 I/O interface channel connector 52 coupled to IBM System/360/370 I/O interface channel port 51. Adapter 36 also has a SCSI bus connector 53. SCSI host 40 has a SCSI device port 54 through which SCSI bus 38 is accessible and which is coupled to SCSI bus connector 53. SCSI host 40 has a communication port 55 coupled to TCP/IP network 42. SCSI host 40 is operable to run software providing an SNA-TCP/IP protocol translator 57 and a fast file transfer unit 58.

Mainframe 16 is operable to communicate SNA and IBM System/360/370 I/O interface channel protocol information via IBM System/360/370 I/O interface channel 18, and SCSI host 40 is operable to communicate SCSI protocol information via SCSI bus 38. According to the teachings of the present invention, adapter 36 is operable to interface SCSI bus 38 with IBM System/360/370 I/O interface channel 18. Adapter 36 communicates information via IBM System/360/370 I/O interface channel 18 and SCSI bus 38 such that adapter 36 appears as a peripheral device to SCSI host 40 and as a system supporting SNA and IBM System/360/370 I/O interface channel protocols to mainframe 16. In this manner, adapter 36 enables bidirectional high bandwidth communication between mainframe 16 and SCSI host 40.

A user of SCSI host 40 can access information housed in mainframe 16 and can provide information to mainframe 16 using the full bandwidth capability of IBM System/360/370 I/O interface channel 18. SNA-TCP/IP protocol translator 57 allows SCSI host 40 to interpret and transmit information according to TCP/IP protocol or SNA protocol. In the embodiment of FIG. 2, TCP/IP protocol translator 57 comprises one version of OC SERVER II software gateway available from OPENCONNECT® SYSTEMS located in Dallas, Tex. Fast file transfer unit 58 allows SCSI host 40 to

move bulk files at optimum speed using IBM System/360/370 I/O interface channel protocols. In the embodiment of FIG. 2, fast file transfer unit 58 comprises one version of OC/GTO software available from OPENCONNECT® SYSTEMS located in Dallas, Tex.

In the embodiment of FIG. 3, adapter 36 comprises a combination hardware and software device contained in a separate housing and operating to interface SCSI bus 38 of SCSI host 40 to mainframe 16 via IBM System/360/370 I/O interface channel 18. Adapter 36 provides IBM System/360/370 I/O interface channel attachment for protocol translator 57 and file transfer unit 58 using a general solution rather than platform-specific hardware and software. Adapter 36 comprises a microcomputer system having electronics and logic components for directly connecting and interfacing SCSI bus 38 with IBM System/360/370 I/O interface channel 18. Adapter 36 can be housed in a small enclosure with connectors for attaching to SCSI bus 38, IBM System/360/370 I/O interface channel 18, and to an external power supply. In another embodiment of the present invention, adapter 36 can provide IBM System/360/370 I/O interface channel attachment for a computer system 40 utilizing IBM System/360/370 I/O interface channel ESCON attachment.

Adapter 36 includes software that works with hardware components to implement low level SNA and IBM System/360/370 I/O interface channel protocols, one or more logical channel devices, such as 3274, SCSI devices to interface with SCSI device drivers on SCSI host 40, administrative functions and glue functions. Adapter 36 also includes firmware that provides a bootstrap loader for downloading operating software from SCSI host 40 across SCSI bus 38. Adapter 36 firmware also provides a driver for a front panel user interface and a configuration program to allow user entry of a SCSI ID, which must be set before SCSI host 40 can contact adapter 36. Additionally, adapter 36 firmware provides power-on self test, diagnostics, and a development/debugging functions.

SCSI host 40 uses system device drivers and hardware at lowest layers to attach to adapter 36. In the embodiment of FIG. 2, SCSI host 40 comprises a UNIX® platform, and protocol translator 57 comprises OC SERVER II. In this configuration, datalink software is used to interface to adapter 36. This software includes an OC SERVER II resident task to implement an application program interface ("API") for adapter 36 and to multiplex program units ("PU's"). Where SCSI host 40 comprises a large computer system, an external UNIX® process may also be used to multiplex OC SERVER II processes. In the embodiment of FIG. 2, fast file transfer unit 58 comprises OC/GTO software. In this environment, changes to a MAMTCP unit in the OC/GTO software are made in order to implement the API for adapter 36.

As a UNIX® platform, SCSI host 40 includes a daemon process that provides error logging capability for adapter 36 either to a file or to an SNMP monitor. This process also provides a path to adapter 36 for administrative utilities residing on SCSI host 40 to allow loading and starting of operating software located in adapter 36, and to provide configuration, dump and trace administration.

Adapter 36 operates to provide numerous functional advantages. Normal DC-interlocked and high-speed transfer features of IBM System/360/370 I/O interface channel 18 can be supported. Data streaming features of IBM System/360/370 I/O interface channel 18 can also be supported as well as burst data rates equal to the 4.5 megabytes per second bandwidth of data streaming on IBM System/360/370 I/O interface channel 18. Adapter 36 operates such that

arbitrary limits to the number of concurrent sessions are not imposed. Available software products including OC SERVER II and OC/GTO can operate in conjunction with adapter 36 with insignificant loss of function. Adapter 36 is also flexible to allow adaptation to implement additional channel protocols over those currently implemented.

Block diagram of adapter components

FIG. 3 is a block diagram of one embodiment of adapter 36 for interfacing SCSI bus 38 with IBM System/360/370 I/O interface channel 18 of FIG. 2. Adapter 36 includes a processor 60 that provides 32-bit CPU, direct memory access ("DMA"), timers, DRAM controller, interrupt controller, parallel ports and serial ports. In the embodiment of FIG. 3, processor 60 comprises an MC68360 microprocessor. Processor 60 is coupled to a SCSI controller 62 and to a channel controller 64 via a bus 61 and respective DMA connections 63. SCSI controller 62 provides an 8-bit CPU bus and a 16-bit DMA bus, and channel controller 64 provides a 32-bit bus. In the embodiment of FIG. 3, SCSI controller 62 comprises an NCR 53CF96-2 SCSI controller, and channel controller 64 comprises an XC4006-6 programmable logic component obtained from XILINX®.

SCSI controller 62 is coupled to a SCSI connector 65 which is operable to connect to SCSI bus 38. Channel controller 64 is coupled to a bus interface 66 and to a tag interface 68. Bus interface 66 and tag interface 68 are coupled to channel connector 69 which is operable to connect to IBM System/360/370 I/O interface channel 18. In the embodiment of FIG. 3, IBM System/360/370 I/O interface channel 18 comprises a block-multiplexed or "bus & tag" channel providing approximately 4.5 megabytes per second of bandwidth. Other embodiments of FIG. 3, System 360/370 I/O interface channel comprise "ESCON" channel providing approximately 17.5 megabytes per second of bandwidth. A dynamic RAM 70, a static RAM 72, a programmable ROM 74, and a front panel 76 are coupled to processor bus 61.

The operation of adapter 36 is controlled by processor 60, SCSI controller 62, and channel controller 64. SCSI controller 62 interfaces with SCSI bus 38 and channel controller 64 interfaces with IBM System/360/370 I/O interface channel 18. Processor 60 links SCSI controller 62 with channel controller 64 to enable direct communication between SCSI bus 38 and IBM System/360/370 I/O interface channel 18. In the embodiment of FIG. 3, the components of adapter 36 can be mounted on and housed in a main circuit board, a connector board, a front panel assembly, an enclosure base with a rear panel, and an enclosure cover. An external power supply, SCSI bus cables, bus/tag converter cables, and terminators can be provided separately.

In the embodiment of FIG. 3, processor 60 of adapter 36 comprises a MOTOROLA® MC68360 and provides a central processing unit ("CPU") and on-board peripherals. An MC68360 implements a 25 MHz CPU32 as its core processor, which is a 32-bit CPU from the 68000 family. As such, processor 60 is fully implemented as a 32-bit machine, and on-board peripherals include DRAM controller, timers, two independent DMA controllers, an interrupt controller, serial ports, baud rate generators, and parallel ports. A serial port resident in processor 60 can be utilized to support a firmware-based debugger program. The serial port can be terminated on a main circuit board of adapter 36 with no provision for a debug port external to the enclosure. An internally accessible ABORT switch can also be provided.

In the embodiment of FIG. 3, SCSI controller 62 of adapter 36 comprises an NCR 53CF96-2. As such, SCSI controller 62 provides an interface to SCSI bus 38 including

line drivers/receivers, and handles all SCSI bus protocol. In the illustrated embodiment, only single-ended drivers and receivers are implemented.

In the embodiment of FIG. 3, channel controller 64 implements drivers and receivers to interface to IBM System/360/370 I/O interface channel 18. Channel controller 64 comprises a programmable logic component available from XILINX® and is programmed to implement a handler for low level IBM System/360/370 I/O interface channel protocol. A 32-byte FIFO in channel controller 64 along with a dedicated DMA channel and interrupt provide adapter 36 real time response capabilities to service IBM System/360/370 I/O interface channel 18.

In the embodiment of FIG. 3, adapter 36 comprises a number of memory components. Dynamic RAM 70 comprises an 8 megabyte, expandable to 32 megabyte, DRAM (2 M×32-bits). Dynamic RAM 70 is operable to provide general user RAM, buffers, and other such functions. Static RAM 72 comprises a 512 kilobyte SRAM (128 K×32-bits) and is operable to provide program storage, stack space and storage for global variables. Programmable ROM 74 comprises a 128 kilobyte EEPROM (64 K×8-bits) and is operable to provide firmware and non-volatile configuration storage. In addition, there is 1536 bytes of system RAM resident in processor 60.

Front panel 76 provides an operator interface for adapter 36 and can comprise an operator panel including LED's, a multiple character LCD display, and membrane switches or buttons. The LED's can be allocated as indicators for power, halt, online, and operational status, and for CPU, IBM System/360/370 I/O interface channel and SCSI bus activity. The membrane buttons can be allocated for RESET, MENU and SELECT functions. In addition to displaying an operational status of adapter 36, the LCD display can be used in conjunction with MENU and SELECT buttons to implement a menuing system for entering configuration data such as SCSI identification code ("SCSI ID").

In the embodiment of FIG. 3, front panel 76 can comprise a modular LED and membrane switch component with a flexible flat cable connector, a two line by sixteen character LCD display for status and configuration information, a ribbon cable, and brackets for mounting to an enclosure base. The LED/switch component provides buttons for RESET, MENU and SELECT, and provides LED indicators for POWER, PROCESSOR HALT and BUSY, CHANNEL and SCSI I/O ACTIVITY, and ONLINE ENABLE and OPERATIONAL OUT CHANNEL STATUS. The LCD display can operate as an 8-bit peripheral off main CPU bus 61. The LCD display can include two lines having an 80-character buffer for storing ASCII data, of which 16 characters are visible at a time. This display can include cursor and highlighting capabilities as well as high level commands for shifting the display. Front panel 76 also includes a reset button hardwired to a board reset circuit of adapter 36. MENU and SELECT buttons can be handled by a parallel port resident in processor 60 with interrupt generating capability. Front panel 76 further includes LED's that are indicators of hardware signals with the exception of ONLINE and PROCESSOR BUSY indicators. ONLINE can be a software-controlled indicator intended to directly reflect a state of the ONLINE setting, entered by an operator via a menuing system for front panel 76. PROCESSOR BUSY can be automatically set by hardware when processor 60 of adapter 36 awakens from a STOP condition, and can be reset by software immediately before issuing a STOP instruction.

All logic components of adapter 36 can reside on a main circuit board mounted to an enclosure base. A DC power

switch and a five-pin circular DIN-connector for DC power can be right-angle components mounted directly to the main circuit board. Post-row connectors for ribbon cables can connect front panel 76. Post-row connectors can also be provided for attaching a debug console and abort switch. The main circuit board can include a 96-pin DIN connector for passing all SCSI and IBM System/360/370 I/O interface channel signals to a connector circuit board.

A connector circuit board can provide mini-D 50-pin connectors 65 for SCSI-In and SCSI-Out for SCSI bus 38, and 100-pin hi-density HIPPI style connectors 69 for Bus/Tag-In and Bus/Tag-Out for IBM System/360/370 I/O interface channel 18. The connector circuit board can mount to a rear panel with connectors protruding through. The connector circuit board can also provide a 96-pin DIN connector for direct attachment to a main circuit board, which supplies electrical signals for SCSI bus 38 and IBM System/360/370 I/O interface channel 18. Such a setup is similar to a VME backplane arrangement utilizing conventional VME parts.

A base of an enclosure for adapter 36 can be a sheet metal component that includes a rear panel for adapter 36. The rear panel can have openings for SCSI-In and SCSI-Out connectors 65, and for Bus/Tag-In and Bus/Tag-Out connectors 69. The rear panel can also have openings for a DC power connector and switch. A circuit board can mount to the rear panel providing connectors 65 for SCSI bus 38 and connectors 69 for IBM System/360/370 I/O interface channel 18. An enclosure cover can be a sheet metal component providing a top and sides for adapter 36, and a bezel for front panel 76.

Adapter 36 can use Bus/Tag-In and Bus/Tag-Out assemblies developed for the Openconnect Systems 3030 hardware gateway 34, to convert from 100-pin high-density HIPPI connectors to standard Bus and Tag serpentine connectors. Cables for attachment to SCSI host 40 can be provided for common SCSI connectors such as mini-D 50 to mini-D 50, mini-D 50 to Centronics, and mini-D 50 to DB-50. An external power supply can be utilized to provide DC power to adapter 36. Connection to adapter 36 can be via a 5-pin circular DIN connector accessible from a rear panel adapter 36. The power supply can comprise a wide range model that needs no special configuration for domestic or international installations, and a country-specific power cord can be provided separately.

Block diagram of software and firmware

FIGS. 4A and 4B are block diagrams of one embodiment of software and firmware operating in SCSI host 40 and adapter 36 of FIGS. 2 and 3. Adapter 36 comprises SCSI transport service 100 which includes a SCSI interface, an application program interface ("APT") provider, eight logical units ("LUN's"), and tape device emulation, as shown. Adapter 36 also comprises adapter control task 102 which includes dispatch, administration, path manager, 3274 manager, and file transfer manager components. Adapter 36 further comprises firmware 104 and channel interface 106. Firmware 104 includes initialization and control, power-on self test, bus interface and firmware services. Channel interface 106 includes channel physical and LDH 3274 and LDH-GTO components. The software and firmware components of adapter 36 are interconnected as shown in FIGS. 4A and 4B.

SCSI host 40 comprises a UNIX® kernel 108 which includes a SCSI interface, tape devices, and a file system interface. SCSI host 40 also comprises an administration daemon 110, a first SNA unit 112, a second SNA unit 114, and a fast file transfer unit 116. In the illustrated embodiment, SNA units 112 and 114 comprise OCSNA

processes, and file transfer unit 116 comprises a MAMTCP (GTO) process established by OC SERVER II software available from OPENCONNECT® SYSTEMS located in Dallas, Tex. The software components of SCSI host 40 are interconnected as shown in FIGS. 4A and 4B.

Functional layers—SCSI bus

SCSI bus 38 comprises a functional layer of adapter 36 and SCSI host 40. The ANSI specification for SCSI, and SCSI-II (ANSI X3.131-1990), defines signals, cables, drivers, connectors, terminators and timings on a SCSI bus. Within this specification, adapter 36 can implement single ended drivers and an 8-bit bus. SCSI host 40 and other daisy-chained peripherals should do the same. Single ended drivers limit the cable length between SCSI devices to six meters. In adapter 36, a SCSI bus interface is implemented by SCSI controller 62, and software is involved in setting configuration parameters such as the SCSI ID. The same is true for SCSI host 40.

Functional layers—SCSI logical

A SCSI logical layer exists in both SCSI host 40 and adapter 36. The SCSI logical layer can operate in a SCSI-I mode of operation until SCSI-II has been negotiated from SCSI host 40. A SYNCHRONOUS DATA TRANSFER REQUEST message is used to negotiate up to 10 MHz synchronous transfers to support data streaming on IBM System/360/370 I/O interface channel 18. The SCSI logical layer can be implemented in adapter 36 by a software component, running mainly at interrupt level, that works with SCSI and DMA controllers in adapter 36. Some functionality can be configurable to accommodate different types of SCSI host computer systems. SCSI logical layer functions on SCSI host 40 can be implemented either in hardware or system device drivers. Some of this functional behavior can be configurable.

The ANSI specification for SCSI defines the logical protocols for operating SCSI bus 38, much of which is dependent on configuration. SCSI devices are classified as Initiators and Targets. With respect to adapter 36, SCSI host 40 is the initiator and adapter 36 is the target. Additional targets as well as initiators may also be present on SCSI bus 38.

At the lower logical level, SCSI implements a finite state machine consisting of "bus phases" which are implemented by various combinations and sequences of control signals. BUS FREE, ARBITRATION, SELECTION and RESELECTION phases are associated with establishing or re-establishing a connection between an initiator and a target to perform an information transfer. COMMAND, DATA (IN or OUT), STATUS, and MESSAGE (IN or OUT) are the information transfer phases.

Device level commands are sent from the initiator to the target during COMMAND phase to perform a device operation or to communicate the desire to enter the DATA phase to transfer data. The SCSI specification defines a Command Descriptor Block ("CDB") and device opcodes for common devices and their operations. When a command completes, the target enters the STATUS phase to report command results to the initiator, then enters MESSAGE phase to send a command complete message. SCSI defines the following status codes: GOOD, CHECK CONDITION, CONDITION MET, BUSY, INTERMEDIATE, INTERMEDIATE CONDITION MET, RESERVATION CONFLICT, COMMAND TERMINATED, and QUEUE FULL.

The MESSAGE phase allows the exchange of control messages between the target and initiator to manage path and data flow. Included are the following: COMMAND COMPLETE, SAVE DATA POINTER, RESTORE

POINTERS, DISCONNECT, INITIATOR DETECTED ERROR, ABORT, MESSAGE REJECT, NO OPERATION, MESSAGE PARITY ERROR, BUS DEVICE RESET, IDENTIFY, and SYNCHRONOUS DATA TRANSFER REQUEST. SCSI defines other messages supporting "linked commands" and "tagged queuing" that are not significant to the illustrated embodiment of adapter 36.

Each SCSI device can define up to eight Logical Units ("LUN's"). As shown in FIGS. 4A and 4B, adapter 36 firmware defines all eight LUN's for potential use since SCSI host 40 may not support dynamic definition of LUN's. To access a LUN in adapter 36, SCSI host 40 arbitrates for SCSI bus 38, selects adapter 36, then issues an IDENTIFY message that specifies a LUN within adapter 36. Some SCSI-I hosts may skip this message and pass the LUN in the CDB. A device command to initiate a data read or a write can then be sent to the LUN in adapter 36. Unless the LUN is ready to proceed with the data transfer, it should respond with a DISCONNECT message to effectively suspend the current command at SCSI host 40 and get off SCSI bus 38.

If the LUN chose to disconnect, then the LUN arbitrates for SCSI bus 38 and perform RESELECTION when it is ready to proceed with the data transfer. The LUN also sends an IDENTIFY message to SCSI host 40 to identify the reconnecting LUN, so that the appropriate pointers will be loaded. This achieves independent operation of the LUN's in adapter 36 and prevents adapter 36 from locking out other devices on SCSI bus 38. It is also possible that application buffering schemes may require that data transfers be interrupted midstream. For these cases, the LUN disconnects as described above, but first issues a SAVE DATA POINTER message to the initiator. The RESTORE POINTERS message is used at RESELECTION time to instruct the initiator to resume the transfer where it left off.

Functional layers—SCSI tape device

SCSI tape devices comprise another layer in SCSI host 40 and adapter 36. Within the ANSI specification for SCSI, chapters are dedicated to discussion of SCSI commands and status, generic SCSI devices and sequential access devices (in chapters 6, 7 and 9, respectively). Provision is also made for vendor-specific operation. All SCSI tape devices should operate within this broad definition.

Device drivers on SCSI host 40 implement an initiator role for supported tape devices. Depending on system implementation, an unrecognized tape device is typically supported by a general SCSI tape driver. This feature is exploited to provide applications access from SCSI host 40 to adapter 36 using the standard file system and system-provided device drivers in UNIX® environments.

SCSI host 40 can issue an INQUIRY command to obtain device information from adapter 36. Adapter 36 responds for all LUN's that adapter 36 is a SCSI-I sequential access device supporting synchronous data transfer (as mentioned above, SCSI-II operation can be negotiated from SCSI host 40 later using the CHANGE DEFINITION command). In UNIX® systems, this causes adapter 36 to be associated with the general SCSI tape driver.

Where SCSI host 40 does not support such a generic SCSI tape device, adapter 36 should include vendor and product identification of a supported tape device in response to the INQUIRY. This may include specification of the platform of SCSI host 40 through front panel 76 of adapter 36 before download occurs.

Asynchronous overlapped input/output ("I/O") is not a universally available feature of tape devices. Therefore, two SCSI tape devices can be included to implement a full-duplex path to adapter 36. Application data can be trans-

ferred to and from adapter 36 using tape WRITE and tape READ commands. Other command handling may be nonproductive, incident to the emulation of a SCSI tape device.

Functional layers—application program interface

An application program interface ("APT") exists in SCSI host 40 for adapter 36. Logical communication paths are created between SCSI host 40 and adapter 36 by defining SCSI tape devices on SCSI host 40 system as shown. Each tape device on SCSI host 40 maps to tape emulation code residing in a LUN on adapter 36, and represents a half-duplex communications path. Two tape devices are used for full-duplex communications.

An application resident on SCSI host 40 can communicate with an application resident on adapter 36 (such as channel devices and administrative components) utilizing one or more logical communication paths and a simple message protocol. Administrative components can be accessed through fixed LUN addresses, 0 and 1. Channel devices can be dynamically configured to reside at any of the remaining LUN address pairs.

Standard file system calls can be used by an application on SCSI host 40 to access a corresponding device resident on adapter 36: open, close, read, write, and select if available. A full-duplex logical path can be opened between SCSI host 40 and adapter 36 by issuing "open" calls to an appropriate pair of tape devices. For a path other than the fixed path to administrative components, the "open" calls can be followed by path control commands to establish the identity of the newly created path. Read and write can be used for passing path control commands, as well as application data, encapsulated in message frames for adapter 36.

A message frame for adapter 36 can be used to maintain message boundaries for application data in a consistent way for all applications. Among other reasons, this allows adapter 36 tape emulation code to present a message-oriented interface to devices on IBM System/360/370 I/O interface channel 18. The message frame can be composed of a Message Header, containing the message length, followed by zero to 64 Kilobytes of application data. Application-defined fields for sub-path identification and command can also be included as part of the Message Header. These fields can help provide a consistent mechanism for applications that require multiplexing capability.

Functional layers—LDH 3274

An LDH 3274 channel device interface comprises another layer in adapter 36. Attachment to IBM System/360/370 I/O interface channel 18 can be implemented using a channel device that emulates the IBM® 3274-41A Control Unit. In FIGS. 4A and 4B, this device is referred to as LDH3274 (Logical Device Handler). LDH3274 manages the connection and the flow of SNA protocol information between mainframe 16 and a physical unit ("PU") configured in adapter 36. There is one LDH3274 instance generated for each PU configured in OCSNA 112 and 114. In adapter 36, SCSI bus 38 separates the LDH(s) from the rest of the IBM System/360/370 I/O interface channel gateway provided by OCSNA 112 and 114. The PU can communicate with the LDH(s) using the message protocol for adapter 36 according to the API for adapter 36.

Since LUN's are limited, a multiplexing scheme can be used for path resolution. A pair of LUN's, corresponding to two tape devices, can be allocated as a full-duplex communications path for multiple PU's/LDH's, possibly from OCSNA processes 112 and 114. Messages flowing over SCSI bus 38 between LDH3274 and OCSNA process 112 or OCSNA process 114 in SCSI host 40 carry a Message

Header for adapter 36 as described above. Multiplexing can be achieved by utilizing a sub-path identifier field in the Message Header for adapter 36. This identifier can be used by a multiplexer process on either side of SCSI bus 38 to route SNA protocol data and control messages to an appropriate destination. Multiplexer processes comprise intermediate software components between the PU's and the LDH's.

One sub-path identifier can be reserved for passing control messages between multiplexer processes for each pair of LUN's configured for an OCSNA process or an LDH3274. The others can be available to be assigned to identify configured LDH's. A separate sub-path for control can allow for the implementation of an efficient flow control mechanism, since flow information for all PU's/LDH's on a LUN pair can be passed in a single message.

A flat BTU buffer, with extra space for header information, can be used to pass SNA protocol data through various SNA layers in adapter 36. This header space in the BTU buffer can be utilized to receive the Message Header for adapter 36, which can be added and stripped away at both ends of SCSI bus 38 without consequence.

The commands listed below are supported by LDH3274 and the "protocol handler". These commands, when appropriate, may be passed over SCSI bus 38 in the command field of a Message Header, and then converted to "ITEM" commands by an LDH interface component. In addition, the flow control mechanism can create a new class of commands exchanged between multiplexing components on either side of SCSI bus 38 that are not seen by the LDH.

PASSBTU	normal message passing
REQMS	request maintenance statistics (no reset)
REQMSR	request maintenance statistics (with reset)
MSRSP	maintenance statistics response
HOSTC	host has sent a connect command
HOSTDC	host has sent a disconnect command
INIZREQ	initialization request from the protocol handler w/ iniz parms
STOPREQ	terminate request from the protocol handler
DIBERROR	invalid dib request from ph
HELLO	are you there? request from the protocol handler
HOWDY	texas type response to a hello
INIZRSP	response to an initialization request
STOPRSP	response to a stop request
SNARFED	ldh is unbelievably messed up
INT	a wake up item from the ph
HOSTCRSP	response to HOSTC
MEMREQ	memory allocation request from protocol handler
MEMRSP	memory allocation response to protocol handler
CLOSEREQ	UNIX terminate request from protocol handler
CLOSERSP	UNIX terminate response to protocol handler
RLSBUF	release buffer
MAKEBUFF	make a channel buffer, attach it to this item, release to channel
HOSTDC_RLS	host sent a disconnect commands plus release this buffer

Functional layers—LDH-GTO

An LDH-GTO channel device interface comprises an additional layer in adapter 36. OC/GTO is a fast file transfer application available from OPENCONNECT® SYSTEMS located in Dallas, Tex. The LDH-GTO comprises a non-SNA protocol communications product that relies on a pair of channel devices to give mainframe applications access to

TCP/IP sockets. A sockets protocol can be passed across IBM System/360/370 I/O interface channel 18 between GTO components residing on mainframe 16 and in SCSI host 40. A GTO channel device protocol can be based on Mitek Access Method ("MAM") which is a channel protocol designed for high-speed data transfer. Consequently, the primary application is for fast file transfer. GTO can be supported in adapter 36 by two channel devices, referred to as Logical Device Handlers ("LDH's"), and a component for interfacing an application in SCSI host 40. One LDH handles inbound data traffic and the other handles the outbound data traffic.

Interface protocol to GTO-LDH can be implemented using the command set listed below.

To LDH:

FFT_SDAREQ	Set device address
FFT_RLSSTAT	Give LDH a status buffer for reporting status
FFT_WRITE	Write data to mainframe (inbound dev)
FFT_RLSBUF	Give LDH a buffer to read data from mainframe (outbound dev)
FFT_CLOSEREQ	Request close on FFT LDH.
FFT_MAXBUFSZ_REQ	Respond with FFT_CLOSERSP
	Tell LDH max buffer size supported

From LDH:

FFT_STAT	Response to FFT_SDAREQ
FFT_SDARSP	LDH's status for FFT device
FFT_WRITE	Write data to mainframe has completed (inbound dev)
FFT_RLSBUF	Read completed, buffer has data from mainframe (outbound dev)
FFT_CLOSERSP	Response to FFT_CLOSEREQ
FFT_MAXBUFSZ_RSP	Response to FFT_MAXBUFSZREQ

These commands flow between the GTO-LDH interface component and the LDH-GTO as ITEM commands over a MOS path. Some originate or terminate at the GTO-LDH interface component. Commands that flow information over SCSI bus 38 are passed in the command field of the Message Header for adapter 36 during tape READ's and WRITE's from application MAMTCP on SCSI host 40, and converted to and from "ITEM" commands by the GTO-LDH interface component.

Functional layers—administrative path

An administrative path for adapter 36 comprises a further layer. A separate administrative path supports logging, tracing, and debugging from SCSI host 40. This path can also be utilized for other administrative operations, such as an initial download for adapter 36, core dumps, local trace buffer dumps, and configuration and control commands to adapter 36. A pair of LUN's (corresponding to two tape devices) is allocated as a full-duplex communications path for handling all administrative traffic to and from adapter 36: an Inbound Admin Device (LUN 0) and the Outbound Admin Device (LUN 1).

Separate software and firmware components exist in adapter 36 and on SCSI host 40 for executing various administrative tasks; such that the tasks can be run concurrently, where practical. As an example, logging, tracing, and a debug session might be run concurrently, whereas a core dump would need to run alone. The sub-path identifier field in the Message Header for adapter 36 can be used to address each Admin component.

Functional layers—bus interface services

Several services resident as firmware on adapter 36 are accessible to SCSI host 40 over one sub-path of the admin-

istrative path. These services include load, dump, debugger and diagnostic services. The debugger includes commands for viewing local traces. These services are collectively referred to as Firmware Bus Interface services.

SCSI host 40 formats a control block (with appropriate Message Header), and writes it to an Inbound Admin Device. When the command completes or when asynchronous character data from one of the services is available, adapter 36 formats a control block (with appropriate Message Header) that can be read by SCSI host 40 over an Outbound Admin Device.

In general, the Firmware Bus Interface services are only available to SCSI host 40 when adapter 36 is under the control of the firmware. Once adapter 36 is loaded and started, these services are unavailable until adapter 36 receives a STOP command and returns to firmware mode.

A LOAD can be used at power up and reset to download operating software into adapter 36. A separate START command can be used to start adapter 36 to operating software. A DUMP can be used at the discretion of SCSI host 40 to retrieve all or part of adapter 36 memory. Diagnostics should be run at power-up or reset. Adapter 36 includes firmware commands to query for previous fatal error status to determine if dump is required before download and to download/update certain firmware components; in particular, logic for channel controller 64.

Functional layers—control path

A control path for adapter 36 comprises a sub-path of Admin, and is used for sending configuration and control commands and data, and for retrieving status information for adapter 36. Adapter 36 can be stopped, via the control path, in order to reload and/or reconfigure. Firmware Bus Interface services are available after a STOP is received, until adapter 36 has been restarted.

Functional layers—logging and tracing

Adapter 36 event messages flow over the logging sub-path of Admin, from adapter 36 to the Admin Daemon on SCSI host 40. A separate tracing path (sub-path of Admin) can be provided to support a comprehensive tracing capability in adapter 36. Trace configuration and start/stop commands can be sent to adapter 36 from SCSI host 40. Trace vectors can be sent from adapter 36 to SCSI host 40 for decode and display or recording in a trace log file.

Functional layers—channel physical interface protocol

A channel physical interface protocol ("CP/IP") comprises the protocol the LDH uses to communicate with channel physical ("CP"). Three control blocks, called channel control areas ("CCA's"), can be passed back and forth between the LDH and CP and contain fields for command, response, and status codes. The control CCA can be used by the LDH to communicate Control Commands to CP and by CP to communicate responses to those commands back to the LDH. The command CCA can be used by the LDH to communicate Interactive Commands to CP and by CP to communicate Interactive Responses back to the LDH. Interactive Commands tell CP to interact with IBM System/360/370 I/O interface channel 18. An asynchronous CCA is used by CP to notify the LDH that an asynchronous event has occurred on IBM System/360/370 I/O interface channel 18.

The LDH passes CCAs to CP by calling a "put" function that processes the CCA within the CP environment while still running as an LDH task. Likewise, CP passes CCAs to the LDH by calling a "receive" function that was passed to CP by the LDH at initialization. The "receive" function may be called within an interrupt service routine as a result of a channel event and then "wake up" the appropriate LDH task to process a CCA. A more complete description of this

interface can be found in the OpenConnect Systems, Inc. Mitek IBM Communications Controller Programming Interface (P/N 350-0142-101) document.

User interface—adapter front panel

Administration of adapter 36 can primarily be accomplished using SCSI host 40. However, certain setup items are administered at adapter 36. Also, adapter 36 includes a capability of reporting status locally, without relying on SCSI host 40 or SCSI bus 38 being operational.

Aside from cabling and power, all local administration for adapter 36 can be performed through front panel 76. Administration includes the reset switch, hardware status LED's, and the configurations/status subsystem. The LED's are allocated as indicators for power, halt, on-line, and Operational-Out status, and for CPU, IBM System/360/370 I/O interface channel and SCSI bus activity indicators.

The configuration/status subsystem is implemented with software and firmware components that interact with a user through two buttons on front panel 76 for "menu" and "select" functions, and a 2-line by 16-character LCD display. Internally, each line has an 80-character capacity. Long lines can be displayed as a marquee display; i.e., continuously scrolling the message through the 16-character window. Normally, the LCD display displays operational status and a last event log message for adapter 36. This is a default state which is reentered after a timeout interval of inactivity for front panel 76.

In the illustrated embodiment, pushing the MENU button on front panel 76 activates a menuing system, displaying a top-level menu on the top display line with the first menu item highlighted, and the cursor in the first character position of the top line. The bottom line displays data associated with the currently highlighted menu item, which is typically another menu. If the top-level menu items do not fit within the 16-character window, each subsequent depression of the MENU button would cause the menu line to scroll to the left by one item. This in turn causes that item to be highlighted, and its associated data to be displayed on the bottom line.

If the bottom display line is a second-level menu, its menu items are displayed with the first item on the line highlighted. Pushing the SELECT button moves the cursor to the bottom line with the first item "selected". Subsequent depressions of the SELECT button cause the cursor to move to the next item if selections fit completely within viewing area; otherwise, second-level menu items to rotate left by one item, and for that item to become "selected". A "selected" second-level menu item is activated by depressing the MENU button. This action also causes the cursor to return to the previous menu. Once a selection has been activated, it remains highlighted and in the first position each time that particular second-level menu is displayed, until a new selection has been activated. An escape ("ESC") option on the second-level menu can be offered to allow return to the previous menu without activating a new selection.

Additional menu nesting can be supported. If a second-level menu item represents another menu heading, then activating that item causes the top line to display the entire "branch" of the currently selected menu items in the "tree", from the top level to the current, and the new menu to display on the bottom line. There is no necessity for limiting the level of nesting that can be achieved using this technique. For menus nested beyond two levels, a TOP option can be offered to allow direct return to the top-level menu.

In the illustrated embodiment, two menu programs are defined. The first runs from the firmware before download. The second runs after adapter 36 is downloaded and started. These menu programs can be identical, except that the

second program does not allow the SCSI ID or Device to be changed and only allows viewing of the current settings.

User interface—administration from SCSI Host

In general, adapter 36 can be administered from SCSI host 40 with a ".INT"-style configuration file, and commands issued through an Admin Client to Admin Daemon 110. The configuration file provides a UNIX device-to-adapter path mapping, download image path, autoload and autostart flags, and identifies a port for Admin Clients to connect up to Admin Daemon 110. Adapter 36 can be loaded and started automatically by starting up Admin Daemon 110 with no arguments and the autoload and autostart flags set in the configuration file. Command line arguments can allow alternate configuration file specification, or override of individual configuration parameters.

A character mode Admin Client, working with Admin Daemon 110, can provide the user interface to adapter 36. In the illustrated embodiment, the following commands are supported:

Dump	dump adapter 36 memory
Load	download adapter 36 operating software
Start	execute adapter 36 operating software
Stop	stop execution of adapter operating software and return to firmware mode
Bugger	invoke the firmware resident debugger mapped to stdin and stdout
Diagnostics	run selected adapter diagnostics
Configure	download adapter 36 configuration
Status	retrieve and display adapter status and configuration
Trace	execute specified trace on adapter
Log	display the event log

Software components—adapter firmware

Firmware 104 is EEPROM-resident code that provides startup routines and low level procedures for interfacing hardware components.

An initialization and control program controls the flow of firmware 104 providing the main line of program execution from power-up or reset. The initialization and control program is responsible for preliminary initialization, dispatching Power-On Self Tests, hardware initialization, installation of interrupt vectors and other firmware components, and initialization of global variables. The firmware control program also sets up a vector table that provides soft linkage to hardware and software components. This allows the downloaded software to access hardware components and firmware entry points without needing hardcoded addresses.

A power-on self test ("POST") program operates to detect the power-on condition and run self tests to verify that hardware components of adapter 36 are functioning properly.

The debugger, or "bugger" for short, is a firmware component with facilities for examining and modifying memory and CPU registers, single-stepping, breakpoints, assembler, disassembler, and other such functions. There are additional facilities for the examination of local component traces for channel and SCSI interfaces, and for running diagnostics. A default debug console is a dumb ASCII terminal attached to a serial port of adapter 36. The bugger runs at interrupt level, driven by keystrokes from the serial port. A firmware command allows the bugger to switch over from the serial port to SCSI bus 38 for its input and output.

A bus interface services firmware component provides administrative and debug services to SCSI host 40 as mentioned above. These services include load and dump capability, as well as access to the bugger and diagnostics.

The bus interface services are started by the firmware control program after all initialization is complete. SCSI devices LUN 0 and 1 are initialized as read and write paths for the bus interface services. The Admin Control component of adapter 36 operating software can later take over LUN's 0 and 1.

A firmware services component provides services to adapter 36 operating software, or other firmware components, to insulate it from the hardware. Included are drivers for the serial port and front panel 76. The firmware services are accessed via trap instruction and processor registers. A serial port driver allows software components to display messages to the debug console. A front panel driver includes a low level and a high level interface. The low level interface allows software components to directly write messages to either line of the LCD display and to independently read and/or reset stored button states. Additionally, the low level interface can allow installation of service routines for each button. The high level interface implements the menu system described above.

The firmware control program installs a menu program to handle the front panel before operating software is downloaded. This program is primarily concerned with configuration parameters to operate the SCSI interface.

SCSI transport service 100 comprises a firmware component that provides the communications pipe between adapter 36 and SCSI host 40. This component is later replaced by a downloaded software component by the same name. SCSI transport service 100 is composed of the three components, a SCSI interface, tape emulation and SCSI transport API which described in more detail below.

Software components—adapter operating Software

Operating software is downloaded to adapter 36 over SCSI bus 38 for the purpose of implementing channel devices. In the illustrated embodiment, software for implementation of 3274 devices comprises OC SERVER II and OC/GTO software available from OPENCONNECT® SYSTEMS located in Dallas, Tex. This downloaded software utilizes OpenConnect Systems, Inc.'s Mitek Operating System ("MOS") for dispatch and interprocess communication services in a multitasking environment.

SCSI transport service 100 comprises a software component that provides the communications pipe between adapter 36-resident applications and SCSI host 40. This service is originally resident in firmware and is subsequently replaced with a downloaded version. SCSI transport service 100 comprises three components as described below.

The first is a SCSI interface and is also referred to as a SCSI protocol handler. The SCSI interface works with SCSI and DMA controllers in hardware to implement the SCSI logical functionality described above. The SCSI interface provides services to the tape emulation component through a defined interface. The SCSI interface comprises interrupt routines to provide real-time response to SCSI bus 38, and SCSI primitive functions for the tape emulation layer. The SCSI interface can operate as either a SCSI-I or SCSI-II target.

A tape emulation component, which is also referred to as the SCSI device handler, emulates eight independent tape devices (LUN 0-7) as described above. Each tape device defines itself to SCSI host 40 as an ANSI standard half-inch nine-track SCSI tape controller. Each device constitutes an independent logical unit with separate state logic and work area to achieve and handle low-level logical functions.

The third component is a SCSI transport API and provides asynchronous, message-oriented Read/Write services to adapter 36-resident applications. The SCSI transport API

component relies on the tape emulation layer to filter out all SCSI activity other than Reads, Writes and Resets. Application-provided routines are dispatched for notification of certain events, such as input/output ("I/O") completion and reset.

Adapter control task 102 comprises a MOS task responsible for interfacing the channel devices and administrative services to SCSI host 40, and for any multiplexing that may be required to implement the interface. Each device type (LDH 3274, LDH GTO) has an interface/manager component residing in adapter control task 102, with MOS path generated to each potential device. The manager components utilize SCSI transport service 100, according to its API, to read and write SCSI bus 38, and convert interrupt events to MOS queue events.

Adapter control task 102 is also home to Admin control and all the administrative functions in adapter 36 that are not provided by firmware. Admin control also uses the SCSI transport service 100 for accessing SCSI bus 38. Additionally, a path management service resides in adapter control task 102 to listen for incoming PATH OPEN commands so that an appropriate LDH Manager can be assigned to a given path. The main line of execution in adapter control task 102 is a dispatch function that responds to MOS queue events to dispatch the appropriate adapter 36 application (such as LDH Manager or Admin).

A MGR3274 component implements the multiplexing protocol described above with respect to LDH 3274 channel device interface. MOS paths are generated to each generated LDH3274 task for routing ITEM's between the multiplexer and each LDH. Routing is based on adapter 36 logical path and the sub-path ID field and command code in adapter 36 Message Header. The routing data is created and managed by MGR3274 in response to path assignments from the path manager, and from LDH OPEN and CLOSE commands from SCSI host 40 applications received over these paths.

MGR3274 can be assigned up to three full-duplex paths (LUN's 2-7), for communicating with SNA processes on SCSI host 40. A buffer pool is allocated for each inbound path to receive transmissions for all LDH's associated with the path. A flow control protocol is executed over each path to achieve logical independence of buffer resources among LDH's. This protocol is also executed over each outbound path to allow the multiplex process running on SCSI host 40 to control its buffer usage.

A GTO-LDH interface provides the LDH interface component described above with respect to the GTO-LDH channel device interface. The GTO-LDH interface is assigned a full-duplex logical path from the Path Manager in response to a PATH OPEN command from MAMTCP running on SCSI host 40. This path is used for communicating status and data across SCSI bus 38; one for the inbound LDH and one for the outbound LDH. A buffer pool is used for receiving inbound transmissions.

Upon initial entry, Admin Control takes over control of LUN's 0 and 1 from the firmware ("F/W") bus interface. The F/W bus interface then becomes one of several end points for routing messages that flow over adapter 36 administrative path. Configuration and status messages can be handled directly by Admin Control.

Adapter control task 102 includes a logging manager and a trace manager as referenced above. The logging manager writes important system events to front panel 76 and to the logging daemon on SCSI host 40.

LDH3274 is a channel device that emulates the IBM 3274-41A Control Unit. Implemented as a MOS task, each generated LDH3274 manages the connection and flow of

SNA data between mainframe 16 and one PU configured in SCSI host 40. The services of channel physical are used to access the Input/Output ("I/O") sub-channel addresses of mainframe 16. MGR3274 in adapter 36 adapter control task 102 is used to present the anticipated interface.

LDH-GTO uses separate sub-channel addresses for inbound and outbound data transfers which are handled by the inbound LDH (ldhbilly) and the outbound LDH (ldhnanny). LDH-GTO devices implement a channel protocol, based on OpenConnect Systems, Inc. Mitek Access Method ("MAM"), for achieving high-speed data transfers. The services of channel physical are used to access the Input/Output ("I/O") sub-channel addresses of mainframe 16. The LDH-GTO interface component in adapter 36 adapter control task 102 presents the anticipated interface. Channel interface

Channel interface 106 includes a Channel Physical which comprises a manager of the IBM System/360/370 I/O interface channel input/output ("I/O") communications for adapter 36. Channel Physical acts as the intermediary between each LDH (e.g., LDH3274 and LDHGTO) and IBM System/360/370 I/O interface channel 18. Channel Physical manages input from the LDH's, schedules work on the channel, and transfers requests and data between the LDH's and IBM System/360/370 I/O interface channel 18.

Communications between IBM System/360/370 I/O interface channel 18 and Channel Physical occurs at adapter 36 channel hardware interface via interrupts and registers. When a channel event occurs, an interrupt invokes a Channel Physical interrupt service routine which in turn reads channel hardware status registers to determine the event and appropriate action. Channel Physical initiates activity on the channel by writing to the appropriate channel hardware registers. The Channel Physical interface to IBM System/360/370 I/O interface channel 18 complies with the specifications in "IBM® System/360 and System/370 I/O interface channel to Control Unit Original Equipment Manufacturers' Information", GA22-6974-08, File No. S360/S370-19. Communications between Channel Physical and the LDH's are achieved through the use of channel physical interface protocol ("CPIP") as mentioned above.

Software components—SCSI host Software

SCSI tape drivers on SCSI host 40 provide access to SCSI bus 38 for applications using adapter 36 as described above. These drivers are supplied by the operating system with platform-dependent installation requirements. The adapter API library is used by UNIX® applications to interface to adapter 36. This library includes calls for creation and deletion of a full-duplex path to adapter 36, as well as calls for passing data to and from adapter 36.

Since UNIX systems, in general, do not support the SELECT function for tape devices and do not honor non-blocking requests, the adapter API library can need to fork separate processes for the read and the write path and utilize shared memory for passing data. Since pipes support SELECT, they may be utilized for control. Pipes may also be considered for passing data, since the implementation can be much simpler than a shared memory implementation.

The SCACPI in first OCSNA process 112 and in second OCSNA process 114 is the data link component and is responsible for interfacing adapter 36 and multiplexing PU's over a full-duplex path to adapter 36. The SCACPI implements multiplexing protocol and is the counterpart to MGR3274 in adapter 36. For SCSI host 40 systems with three or fewer concurrent OCSNA processes, SCACPI can utilize the adapter API library for access to adapter 36. For large systems, SCACPI can be configured to go through an

OCSNA MUX process. SCACPI coexists with other data link components in the data link task by utilizing a new multiplexing layer that interfaces the NUC. SCACPI may be implemented as a single data link task.

The OCSNA MUX process is a separate UNIX® process responsible for multiplexing LDH3274 traffic from multiple OCSNA processes over a pair of SCSI tape devices. The OCSNAMUX process is used for systems running more than three concurrent OCSNA processes through adapter 36 (or GTO and more than two concurrent OCSNA processes). An API utilizing shared memory and PIC is defined to interface the OCSNA processes 112 and 114. The multiplexing protocol described above with respect to the LDH 3274 channel device interface is used to access MGR3274 in adapter 36.

MAMTCP is a SCSI host 40 workstation application that provides "sockets" access to mainframe applications.

Communication is achieved using reads and writes over the full-duplex path provided by the adapter API library. Adapter 36 Message Header is utilized to multiplex input/output and control functions with data using only the read and write functions.

The Admin Daemon 110 is a UNIX® process that provides the path for all adapter 36 administration from SCSI host 40 using the adapter API. Several administrative functions can be provided directly by Admin Daemon 110, including load, start and event logging. Admin Daemon 110 parses command line arguments and a configuration file for its base input. In addition, Admin Client sessions are supported over socket connections for initiating additional administrative operations. Admin Daemon 110 also provides a SNMP agent component to support certain administrative functions from a SNMP manager.

Admin clients connect to Admin Daemon 110 over socket connections for manual or interactive adapter 36 administration. Admin Daemon 110 provides an ASCII test command set. This allows a generic Admin client to be developed that supports all Admin operations by simply mapping STDIN and STDOUT to Admin Daemon 110 connection. In addition, this client could provide a GUI interface to modify the configuration file. Shell scripts invoking the generic Admin Client can be developed to provide individual Admin utility functions such as "load" and "dump".

Buffers and Data Flow

With respect to outbound SNA data flow, on adapter 36, each LDH3274 allocates a pool of buffers for receiving outbound data from IBM System/360/370 I/O interface channel 18. As outbound data is received into these buffers, they are sent to MGR3274 to be sent across SCSI bus 38. They are held up in MGR3274 until a READ has been received from the outbound tape device, and the flow control mechanism indicates that the corresponding PU can receive the data. At this time, the data can be sent across SCSI bus 38 and the buffer is subsequently released to the LDH. If the LDH runs out of buffers, the channel enters a "slowdown" mode.

On SCSI host 40 system, SCACPI in OCSNA process 112 or 114 allocates a single pool of buffers to receive transmissions from adapter 36 outbound to all PU's in OCSNA process 112 or 114. SCACPI tries continuously to read the outbound tape device, utilizing the "select" function to avoid blocking. As outbound data is received, the buffers are routed to the appropriate PU, and eventually released back to SCACPI. A flow control mechanism between SCACPI in OCSNA process 112 or 114 and MGR3274 on adapter 36 prevents any single PU from using more than its share of buffers.

With respect to inbound SNA data, on SCSI host 40, SCACPI within OCSNA process 112 or 114 receives buffers of inbound data from PUs. The source application and owner of the buffer can be irrelevant. If the inbound path can be written without blocking, and the flow control mechanism indicates that the corresponding LDH can receive the data, then the write operation proceeds causing data to be transmitted over SCSI bus 38. The buffer is then released back to its owner. Each source application is responsible for handling the case of inbound buffer depletion.

On adapter 36, MGR3274 allocates a single pool of buffers for all the LDH's associated with a given OCSNA process 112 or 114. As long as buffers are available, MGR3274 attempts to read the inbound path for that OCSNA process 112 or 114. As inbound data is received, the buffers are routed to the appropriate LDH for transmission over IBM System/360/370 I/O interface channel 18, and eventually released back to MGR3274.

The same flow control mechanism used for outbound transmissions is used for inbound transmissions, preventing any single LDH from using more than its share of buffers. For both directions of flow, adapter 36 Message Protocol allows transmissions to be blocked-up over SCSI bus 38 for improved performance.

With respect to file transfer data, on adapter 36, the GTO-LDH interface component in adapter control task 102 allocates all the buffers. GTO-LDH interface keeps some for inbound transmissions and passes some for outbound transmissions. On SCSI host 40, MAMTCP allocates a buffer pool to be used for both channel and TCP/IP.

Outbound data is received by ldhnanny who passes the buffer to GTO-LDH interface. Outbound buffers are held up here until a READ has been received from the outbound tape device. At this time, the data can be sent across SCSI bus 38 and the buffer subsequently released to the LDH. If the LDH runs out of buffers, the channel enters a "slowdown" mode. As long as buffers are available, MAMTCP on SCSI host 40 attempts to continuously read the outbound path, utilizing the "select" function to avoid blocking.

MAMTCP writes inbound data to the inbound path when the "select" function indicates that blocking can not occur, causing data to be transmitted over SCSI bus 38. As long as buffers are available, GTO-LDH interface attempts to read the inbound path. As inbound data is received, the buffers are passed for transmission over the channel, and eventually released back to GTO-LDH interface.

A single SCSI host 40 may require more than one adapter 36 to address performance issues, fault tolerance, or to connect to multiple mainframes. This can be accomplished by setting up SCSI host 40 with additional tape devices at a different target address. This is platform dependent. SCSI host 40 applications are started with the appropriate device names to direct them to the correct target adapter and LUN's. The only special requirements for handling more than one adapter 36 are associated with Admin Daemon 110 and its configuration file. An Admin Daemon 110 should be started for each attached adapter 36 with a command line parameter specifying its unique configuration file. Each configuration file should specify the device mapping for adapter 36, and provide a unique port address for client connections to Admin Daemon 110.

Schematics for adapter

FIGS. 5A, 5B-1, 5B-2, 5B-3, 5B-4, 5C-1, 5C-2, 5D, 5E-1, 5E-2, 5F-1, 5F-2, 5G, 5H-1, 5H-2, 5H-3, 5I, 5J-1, 5J-2, 5J-3 and 5K are schematics showing one embodiment of the components and interconnections for adapter 36 of FIGS. 2 through 4. FIG. 5A shows a list of building materials and a cross-reference chart.

FIGS. 5B-1, 5B-2, 5B-3 and 5B-4 are schematics showing processor 60 of FIG. 3 including a number of devices interconnected as shown. Processor 60 comprises a 68360 QUICC, which is a MOTOROLA® 32-bit processor, and functions as the central processing unit for adapter 36. Device 120 comprises a 25 MHz crystal oscillator and operates to provide timing signals. Device 122 comprises an address decoder and operates to provide additional capability from the central processor. Device 124 comprises an RS232 interface chip and provides signal conversion from RS232 plus/minus 12 volt signals to standard TTL CMOS signals which are used by processor 60.

FIGS. 5C-1, 5C-2 are schematics showing memory facilities used by processor 60 including a number of devices interconnected as shown. Devices 126 comprise static random access memory chips, each holding 32-bits×128 K for a total of 512 kilobytes of memory storage space. Device 128 comprises a single inline memory slot which can hold from 4 megabytes to 64 megabytes of dynamic random access memory. Device 130 comprises a flash EEPROM, and device 132 comprises a driver circuit for device 130. Device 130 is operable to store debugger code, initialization code and various other boot code. Device 130 also maintains the XYLINK® image used at power-up to program channel controller 64 which comprises an XC4006-6 programmable gate array from XILINX®.

FIG. 5D is a schematic showing SCSI controller 62 of FIG. 3 including a number of devices interconnected as shown. SCSI controller 62 provides an interface between processor 60 and SCSI bus 38. In the illustrated embodiment, SCSI controller 62 comprises an NCR 35CF96-2.

FIGS. 5E-1, 5E-2 are schematics showing a number of devices interconnected as shown. A boundary scan connector 134 is shown and is operable to provide testing. A reset connector 136 is shown and operates as an internal reset generator used for testing during manufacturing. An LCD interface 138 is shown and operates to drive the LCD display of front panel 76. A front panel interface 140 is shown and operates to provide a connector to the buttons and LED's of front panel 76. Device 142 comprises an LED driver. Device 144 comprises a four-bit counter. Device 146 comprises an input/output ("I/O") controller. Together device 144 and device 146 operate as a state machine for the LCD and idle LED on the front panel 76.

FIGS. 5F-1, 5F-2 are schematics showing channel controller 64 of FIG. 3 including a number of devices interconnected as shown. FIGS. 6A-1, 6A-2, 6B, 6C-1, 6C-2, 6D-1, 6D-2, 6E-1, 6E-2, 6F-1, 6F-2, 6G-1, 6G-2, 6H, 6I-1, 6I-2, 6J-1, 6J-2, 6K-1, 6K-2, 6L-1, 6L-2, 6M-1, 6M-2, 6M-3, 6M-4, 6N-1, 6N-2, and 6O-1, 6O-2, as described below, are schematics showing the internal logic components and interconnections of channel controller 64. Channel controller 64 comprises a programmable gate array from XILINX®. Device 148 comprises a 256×4-bit SRAM used for address decoding and channel selection addressing. Device 150 comprises an 18 MHz crystal.

FIG. 5G is a schematic showing the structure of bus interface 66 of FIG. 3. Device 152 comprises a parity generator. Device 154 comprises a 10-bit register. Devices 156 comprise tristate registers operable to control whether or not data is going out bus interface 66 or data is looped back for testing purposes. Device 158 comprises a parity generator that is used to check for bus-out parity. Devices 160 comprise physical signal drivers going out to IBM System/360/370 I/O interface channel 18 and operating physically to drive IBM System/360/370 I/O interface channel bus 18.

FIGS. 5H-1, 5H-2, 5H-3 are schematics showing tag interface 68 of FIG. 3 including a number of devices interconnected as shown. In general, FIG. 3 bus interface 66 provides a data interface between channel controller 64 and IBM System/360/370 I/O interface channel 18, and tag interface 68 provides control signals between channel controller 64 and IBM System/360/370 I/O interface channel 18. Device 162 comprises a voltage reference device used to generate a PRST signal and a reset hard signal. Devices 164 comprise physical drivers for the tag interface bus. Device 166 comprises a loop-back generator that is used for testing similar to that described with respect to bus interface 66. Devices 168 comprise receivers which receive control signals from IBM System/360/370 I/O interface channel 18. Device 170, device 172 and device 174 are operable for reset timing. Device 170 is a programmable component which is operable for selection control for routing signals to and from IBM System/360/370 I/O interface channel 18. Device 176 comprises a selection circuit which functions in the event of power loss to replace device 170 and device 164 with two hardware relays.

FIGS. 5I and 5J-1, 5J-2, 5J-3 are schematics showing SCSI connector 65 and channel connector 69 of FIG. 3 including a number of devices interconnected as shown. FIGS. 5J-1, 5J-2, 5J-3 show a connector board that is connected to the main board through a connector shown in FIG. 5L. The left side of the FIG. 5J-1 shows the bus and tag connector. Device 180 comprises an interface to device 178 of FIG. 5L. Devices 182 comprise connectors to SCSI bus 38. Devices 184 comprise two relays which are driven by device 176 and are used for selection.

FIG. 5K is a schematic showing a number of components and interconnections. Device 186 provides power to the main board. Device 188 is a power switch. The combination of the capacitive array 190 and the beads 192 construct an LC network. The large number of capacitors are by-pass capacitors which supply clean voltages for the channel controller 64 and a phase lock loop inside processor 60. The large number of resistors 194 are all pull-up resistors and the components 196 are spares.

Schematics for channel controller

FIGS. 6A-1, 6A-2, 6B, 6C-1, 6C-2, 6D-1, 6D-2, 6E-1, 6E-2, 6F-1, 6F-2, 6G-1, 6G-2, 6H, 6I-1, 6I-2, 6J-1, 6J-2, 6K-1, 6K-2, 6L-1, 6L-2, 6M-1, 6M-2, 6M-3, 6M-4, 6N-1, 6N-2, and 6O-1, 6O-2 are schematics showing one embodiment of logic components and interconnections for channel controller 64.

FIGS. 6A-1, 6A-2 are schematics showing address decode including a number of logic components interconnected as shown. Devices 200 of FIG. 6A comprise an inputs of channel controller 64. Channel controller 64 comprises a XILINX® XC4006-6 which is a RAM-based field programmable gate array having an SRAM on board. At power-up, an image is brought from programmable ROM 74 and loaded into SRAM on board channel controller 64. Devices 202 of FIGS. 6A-1, 6A-2 are schematic outputs of channel controller 64. Channel controller 64 has bi-directional pins as well. The address decode logic shown in FIGS. 6A-1, 6A-2 include a device 204 which comprises a state machine. Generally, address decode takes a 256 byte address and separates the address into smaller width addresses, typically an 8-bit, 16-bit or 32-bit address.

FIG. 6B is a schematic showing command registers including a number of logic components interconnected as shown. Register 206 is used to control loop back diagnostics. Command register 208 is written only once at initialization. For example, device 210 has the interrupt allow bit

which indicates there should be an interrupt after the next sequence of operations is completed. Device 212 is an online bit which tells the rest of the system that the controller is on-line. Command register 214 and 216 store similar command bits.

FIGS. 6C-1, 6C-2 are schematics showing the selection controller including a number of logic components interconnected as shown. The selection controller is used in two different circumstances. The first circumstance is during the poll situation where some entity is requesting use of IBM System/360/370 I/O interface channel 18. The selection can be characterized as mainframe 16 alerting processor 60 that some action needs to be taken. The selection controller is also used in the circumstance where the selection is not intended for processor 60, and channel controller 64 functions to propagate two signals which comprise the selection down IBM System/360/370 I/O interface channel 18. The selection controller is also used to create a busy signal where a device can tell mainframe 16 via IBM System/360/370 I/O interface channel 18 that the device is busy. The busy signal is created, stored and generated from device 218 of FIGS. 6C-1, 6C-2.

FIGS. 6D-1, 6D-2 are schematics showing the tag sequencer including a number of logic components interconnected as shown. The tag sequencer operates to control the sequence of signalling that happens over tag interface 68. Tag interface 68 provides an interface that passes control signals between mainframe 16 and processor 60. The channel protocol requires particular signals to go high and low at various times within the passing of control information. FIGS. 6D-1, 6D-2 are schematics showing one embodiment of a hardware solution to tag sequencing.

FIGS. 6E-1, 6E-2 are schematics showing the start and termination controller including a number of logic components interconnected as shown. The start and termination controller provides intermediate states, especially the start state which is between the selection from mainframe 16 and the beginning of data transfer. The start and termination controller also provides similar signalling prior to termination of the entire order with the termination signals.

FIGS. 6F-1, 6F-2 are schematics showing the byte clocks and "go" tag sequencer including a number of logic components interconnected as shown. The byte clock is used internally to physically request IBM System/360/370 I/O interface channel 18 to transfer data to processor 60 or to transfer data to channel 18 from processor 60 one byte at a time. The go tag signal occurs between the start process and the termination process and remains on while data transfer is taking place. The upper righthand corner of FIG. 6F-2 is a clock divider for the 18 MHz crystal signal.

FIG. 6G-1, 6G-2 are schematics showing the data transfer controller including a number of logic components interconnected as shown. The data transfer controller operates under two different types of data transfer. One type is synchronous, and the other type is asynchronous. The data-in and service-in lines and the service-out and data-out lines are used in both types of data transfer. In synchronous data transfer, which is referred to as data streaming, the data-in and service-in lines are used alternatively to transfer data into IBM System/360/370 I/O interface channel 18. At some point after that process begins, the service-out and data-out lines are actuated by mainframe 16 to indicate that synchronous data transfer was received. In the asynchronous mode of transfer, the data-in and service-in lines are used to request a data transfer, and the service-out and data-out lines are then used to acknowledge the request. The propagation delay in IBM System/360/370 I/O interface channel 18,

itself, actually reduces the transfer rate in the asynchronous mode because processor 60 must wait for acknowledgement on the service-out and data-out lines before data transfer can take place.

FIG. 6H is a schematic showing the error interrupt control including a number of logic components interconnected as shown. The error interrupt control operates to control three conditions from mainframe 16: resets, selective reset and interface disconnect. Error interrupt control also has three internally generated error conditions. The first one is a parity error which is a standard check on data. If the data is corrupted for some reason, the data transfer must be stopped and re-started. Another error is a switch transition that is an online/offline switch. If this switch has been actuated, processor 60 decides whether or not to go online or whether to go offline based on the current status so that data transfer in process is not interrupted. Data streaming/time out is a counter that times the interval between responses from mainframe 16 to ensure that if IBM System/360/370 I/O interface channel 18 is inactive for too long, processor 16 disconnects to ensure IBM System/360/370 I/O interface channel 18 is not reserved by a nonfunctioning entity.

FIGS. 6I-1, 6I-2 are schematics showing the status interrupt controller including a number of logic components interconnected as shown. One function on the righthand side of FIG. 6I is a physical interrupt signal that goes to processor 60. The status interrupt controller generates a variety of status signals which are used by processor 60.

FIGS. 6J-1, 6J-2 and 6K-1, 6K-2 are schematics showing the processor data multiplexers providing an interface between the 32-bit data bus that connects processor 60 and channel controller 64 including a number of logic components interconnected as shown. The processor data multiplexers operate both for input and output of data on that bus.

FIGS. 6L-1, 6L-2 are schematics showing the bus-in and bus-out for the connection of data lines from bus interface 66 to channel controller 64 including a number of logic components interconnected as shown. Bus-in on the right side of FIGS. 6L-1, 6L-2 are schematics comprising the output of channel controller 64 to mainframe 16. Bus-out comprises the data coming out of mainframe 16 and going into channel controller 64. Devices 220 comprise an address multiplexer for the 256x4 SRAM shown as device 148 of FIGS. 5F-1, 5F-2. Devices 220 comprise address modes for the SRAM, and card 0, 1, 2, 3 comprises the data mux for the SRAM.

FIGS. 6M-1, 6M-2, 6M-3, 6M-4 are schematics showing a data interchange and FIFO system including a number of logic components interconnected as shown. A data path 222 from IBM System/360/370 I/O interface channel 18 is shown in the upper lefthand corner of FIG. 6M-1. Below that, a data path 224 from processor 64 is shown. A data path 226 to processor 64 is shown in the upper righthand corner, and a data path 227 to the channel is shown on the lower right side of FIGS. 6M-2, and 6M-4. Data is received from processor 64 in a 32-bit wide data path and is received from IBM System/360/370 I/O interface channel 18 in an 8-bit wide data path. The data is first placed in a parallel array 228 of data input and multiplexers used to load input registers 230 in the center of FIG. 6M-1. The input registers 230 are loaded in parallel if the data width is 32-bits or sequentially if the data width is 8 bits. The input registers 230 are then used to load an SRAM 232. The SRAM 232 is addressed in a FIFO configuration to load the output registers 234. The output registers 234 are then routed either to processor 64 or to IBM System/360/370 I/O interface channel 18 depending upon the desired destination. At the bottom of FIG. 6M is shown logic 236 which is the channel positioning controller.

The channel positioning controller logic 236 controls which of the input registers 230 is being loaded and which of the output mixers are being accessed for output to IBM System/360/370 I/O interface channel 18.

FIGS. 6N-1, 6N-2 are schematics showing the FIFO push-pop controller including a number of logic components interconnected as shown. This circuit creates control signals that control whether or not data is being pushed into the FIFO or popped out of the FIFO and that control in what order both operations are occurring.

FIGS. 6O-1, 6O-2 are schematics showing the FIFO address and flag controller system including a number of logic components interconnected as shown. The center part of FIGS. 6O-1, 6O-2 are the push address counter and the pop address counter which is the address control for the FIFO. The FIFO flag controller at the bottom of FIGS. 6O-1, 6O-2 determine whether or not the FIFO is empty or full. Direct memory access ("DMA") request is a line that is pulled by channel controller 64 any time channel controller 64 needs data or has data to be placed in memory using a DMA operation.

Overview

An adapter for interfacing a SCSI bus with an IBM System/360/370 I/O interface channel constructed according to the teachings of the present invention provides bidirectional communication at high data bandwidth to take advantage of the bandwidth of the IBM System/360/370 I/O interface channel. Because of the wide availability of SCSI device ports on computer workstations and personal computers, the adapter of the present invention benefits numerous information systems currently used by organizations that include both SNA and TCP/IP environments.

An adapter constructed according to the teachings of the present invention operates to provide numerous functional advantages. Normal DC-interlocked and high-speed transfer features of the IBM System/360/370 I/O interface channel can be supported. Data streaming features of the IBM System/360/370 I/O interface channel can also be supported as well as burst data rates equal to the bandwidth of data streaming on the IBM System/360/370 I/O interface channel. The adapter operates such that arbitrary limits to the number of concurrent sessions are not imposed. The adapter is also flexible to allow adaptation to implement additional channel protocols.

Although the present invention has been described in detail, it should be understood that various changes, substitutions and alterations can be made hereto without departing from the spirit and scope of the invention as defined by the appended claims.

What is claimed is:

1. A data processing system including a plurality of computer systems, the data processing system comprising:
 - a first computer system having an IBM System/360/370 I/O interface channel, the first computer system operable to communicate SNA and non-SNA protocol information via the IBM System/360/370 I/O interface channel;
 - a second computer system having a SCSI bus, the second computer system operable to communicate SCSI protocol information via the SCSI bus; and
 - an adapter coupled to the IBM System/360/370 I/O interface channel of the first computer system and the SCSI bus of the second computer system, the adapter operable to interface the SCSI bus with the IBM System/360/370 I/O interface channel to allow bidirectional communication between the IBM System/360/370 I/O interface channel of the first computer system and the SCSI bus of the second computer system.

2. The data processing system of claim 1, wherein the adapter is further operable to define a plurality of logical units representing SCSI devices for communicating information via the SCSI bus.

3. The data processing system of claim 2, wherein the plurality of logical units represent tape devices.

4. The data processing system of claim 1, wherein the adapter comprises:

a SCSI interface unit, the SCSI interface unit coupled to the SCSI bus;

a channel interface unit, the channel interface unit coupled to the IBM System/360/370 I/O interface channel; and

a processor coupled to the SCSI interface unit and to the channel interface unit, the processor operable to control operation of the adapter.

5. The data processing system of claim 4, wherein the SCSI interface unit comprises a dedicated SCSI controller.

6. The data processing system of claim 4, wherein the channel interface unit comprises a programmable logic device.

7. The data processing system of claim 4, wherein the processor comprises a 32-bit microprocessor.

8. The data processing system of claim 1, wherein the first computer system comprises an IBM or compatible main-frame computer system.

9. The data processing system of claim 1, wherein the second computer system comprises a UNIX-based computer workstation.

10. The data processing system of claim 1, wherein the second computer system comprises a personal computer.

11. The data processing system of claim 10, wherein the second computer system operates under a WINDOWS™ operating system.

12. An adapter for interfacing a SCSI bus with an IBM System/360/370 I/O interface channel, the adapter comprising:

a channel interface unit operable to couple to an IBM System/360/370 I/O interface channel, the channel interface unit operable to communicate SNA and non-SNA protocol information via the IBM System/360/370 I/O interface channel;

a SCSI interface unit operable to couple to a SCSI bus, the SCSI interface unit operable to communicate SCSI protocol information via the SCSI bus; and

a processor coupled to the channel interface unit and to the SCSI interface unit, the processor operable to control the channel interface unit and the SCSI interface unit to allow bidirectional communication between the SCSI bus and the IBM System/360/370 I/O interface channel.

13. The adapter of claim 12, wherein the processor is further operable to define a plurality of logical units representing SCSI devices for communicating information via the SCSI bus.

14. The adapter of claim 13, wherein the plurality of logical units represent tape devices.

15. The adapter of claim 12, wherein the channel interface unit, the SCSI interface unit and the processor are coupled to a processor bus.

16. The adapter of claim 15, further comprising a front panel coupled to the processor bus and operable to provide a user interface.

17. The adapter of claim 15, further comprising a dynamic RAM, a static RAM, and a programmable ROM, each coupled to the processor bus.

18. The adapter of claim 12, wherein the SCSI interface unit comprises a dedicated SCSI controller.

19. The adapter of claim 12, wherein the channel interface unit comprises a programmable logic device.

20. The adapter of claim 12, wherein the processor comprises a 32-bit microprocessor.

21. An adapter for interfacing a SCSI bus with an IBM System/360/370 I/O interface channel, the adapter comprising:

a channel connector, the channel connector operable to couple to an interface channel port of a first computer system, the interface channel port providing access to an IBM System/360/370 I/O interface channel;

a bus and tag interface coupled to the channel connector; a channel controller coupled to the bus and tag interface, the channel controller operable to communicate SNA protocol information via the IBM System/360/370 I/O interface channel;

a SCSI connector, the SCSI connector operable to couple to a SCSI device port of a second computer system, the SCSI device port providing access to a SCSI bus;

a SCSI controller coupled to the SCSI connector, the SCSI controller operable to communicate SCSI protocol information via the SCSI bus;

a processor bus coupled to the channel controller and to the SCSI controller;

a user interface coupled to the processor bus;

a dynamic RAM device coupled to the processor bus;

a static RAM device coupled to the processor bus;

a programmable ROM coupled to the processor bus; and

a processor coupled to the processor bus, the processor operable to manage operation of the channel controller and the SCSI controller to allow bidirectional communication between the SCSI bus and the IBM System/360/370 I/O interface channel.

22. The adapter of claim 21, wherein the SCSI controller comprises an NCR35CP96-2 SCSI controller.

23. The adapter of claim 21, wherein the channel controller comprises a programmable logic device.

24. The adapter of claim 21, wherein the processor comprises a MC68360 32-bit microprocessor.

25. The adapter of claim 21, wherein the dynamic RAM device comprises an eight megabyte DRAM.

26. The adapter of claim 21, wherein the static RAM device comprises a 512 kilobyte SRAM.

27. The adapter of claim 21, wherein the programmable ROM device comprises a 128 kilobyte EEPROM.

28. The adapter of claim 21, wherein the user interface comprises a front panel having LED's, buttons and an LCD display.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 5,640,541
DATED : June 17, 1997
INVENTOR(S) : Bartram, *et al.*

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby
corrected as shown below:

Title page, [73], delete "Openconnect Systems, Inc." and insert
-- Openconnect Systems Incorporated --.

Signed and Sealed this
Seventeenth Day of March, 1998

Attest:



BRUCE LEHMAN

Attesting Officer

Commissioner of Patents and Trademarks



US006141731A

United States Patent [19][11] **Patent Number:** **6,141,731****Beardsley et al.**[45] **Date of Patent:** **Oct. 31, 2000**[54] **METHOD AND SYSTEM FOR MANAGING DATA IN CACHE USING MULTIPLE DATA STRUCTURES**[75] **Inventors:** **Brent Cameron Beardsley; Michael Thomas Benhase; Douglas A. Martin; Robert Louis Morton; Mark A. Reid,** all of Tuscon, Ariz.[73] **Assignee:** **International Business Machines Corporation, Armonk, N.Y.**[21] **Appl. No.:** **09/136,630**[22] **Filed:** **Aug. 19, 1998**[51] **Int. Cl.⁷** **G06F 12/00**[52] **U.S. Cl.** **711/136; 711/133; 711/135; 711/159; 711/160**[58] **Field of Search** **711/133, 135, 711/136, 159, 160**[56] **References Cited****U.S. PATENT DOCUMENTS**

4,437,155 3/1984 Sawyer et al. .
 4,458,316 7/1984 Fry et al. .
 4,468,730 8/1984 Dodd et al. .
 4,489,378 12/1984 Dixon et al. .
 4,490,782 12/1984 Dixon et al. .
 4,533,995 8/1985 Christian et al. .

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

0674263 2/1995 European Pat. Off. .
 6052060 3/1994 Japan .

OTHER PUBLICATIONS

Stone et al., "Optional Partitioning of Cache Memory", 1992, pp. 1054-1068.
 Improving Most Recently User Change Prefetching, IBM Technical Disclosure Bulletin, vol. 36, No. 08, Aug. 1993.
 Optimized Look-Ahead Extension on Sequential Access, IBM Technical Disclosure Bulletin, vol. 39, No. 11, Nov. 1996.

Direct Access Storage Device Cache Segment Management, IBM Technical Disclosure Bulletin, vol. 37, No. 08, Aug. 1994.

Cache System for Hard Disk System Utilizing the Access Data Address, IBM Technical Disclosure Bulletin, vol. 38, No. 01, Jan. 1995.

Direct Memory Access Paging and Remote DRAM Access Through an Optimized Memory Mapping Mechanism, IBM Technical Disclosure Bulletin, vol. 38, No. 06, Jun. 1995.

Non-Volatile Cache Storage Allocation Algorithm, IBM Technical Disclosure Bulletin, vol. 38, No. 12, Dec. 1995.

Fixed Storage Allocation of Input-Output Buffers, IBM Technical Disclosure Bulletin, vol. 39, No. 03, Mar. 1996.

Remote Copy Administrator's Guide and Reference, DFSMS/MVS Version 1, Third Edition, Jul. 1996, IBM Doc. No. SC35-0169-02.

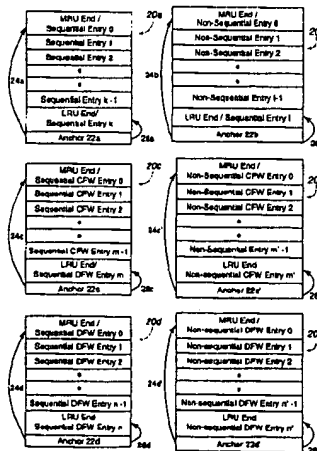
(List continued on next page.)

Primary Examiner—John W. Cabeca**Assistant Examiner**—Brian R. Peugh**Attorney, Agent, or Firm**—David W. Victor; Konrad Raynes & Victor LLP

[57]

ABSTRACT

Disclosed is a cache management scheme using multiple data structure. A first and second data structures, such as linked lists, indicate data entries in a cache. Each data structure has a most recently used (MRU) entry, a least recently used (LRU) entry, and a time value associated with each data entry indicating a time the data entry was indicated as added to the MRU entry of the data structure. A processing unit receives a new data entry. In response, the processing unit processes the first and second data structures to determine a LRU data entry in each data structure and selects from the determined LRU data entries the LRU data entry that is the least recently used. The processing unit then demotes the selected LRU data entry from the cache and data structure including the selected data entry. The processing unit adds the new data entry to the cache and indicates the new data entry as located at the MRU entry of one of the first and second data structures.

40 Claims, 7 Drawing Sheets

U.S. PATENT DOCUMENTS

4,583,166 4/1986 Hartung et al. .
 4,603,382 7/1986 Cole et al. .
 4,636,946 1/1987 Hartung et al. .
 4,875,155 10/1989 Iskiyan et al. .
 4,882,642 11/1989 Tayler et al. .
 4,956,803 9/1990 Tayler et al. .
 4,979,108 12/1990 Crabbe, Jr. .
 5,134,563 7/1992 Tayler et al. .
 5,263,145 11/1993 Brady et al. .
 5,297,265 3/1994 Frank et al. .
 5,394,531 2/1995 Smith 711/136
 5,426,761 6/1995 Cord et al. .
 5,432,919 7/1995 Falcone et al. .
 5,432,932 7/1995 Chen et al. .
 5,434,992 7/1995 Mattson .
 5,440,686 8/1995 Dahman et al. .
 5,440,727 8/1995 Bhide et al. .
 5,446,871 8/1995 Shomler et al. .
 5,481,691 1/1996 Day, III et al. .
 5,504,861 4/1996 Crockett et al. .
 5,542,066 7/1996 Mattson et al. 711/136
 5,551,003 8/1996 Mattson et al. .
 5,574,950 11/1996 Hathorn et al. .
 5,590,308 12/1996 Shih .
 5,592,618 1/1997 Micka et al. .
 5,606,688 2/1997 McNutt et al. .
 5,615,329 3/1997 Kern et al. .
 5,623,599 4/1997 Shomler .
 5,623,608 4/1997 Ng .
 5,627,990 5/1997 Cord et al. .
 5,636,359 6/1997 Beardsley et al. .
 5,651,136 7/1997 Denton et al. .
 5,692,152 11/1997 Cohen et al. 711/140

5,805,855 9/1998 Liu 711/108
 6,041,390 3/2000 Liu et al. 711/110

OTHER PUBLICATIONS

Pinter and Yoaz; Tango: a Hardware-based Data Prefetching Technique for Superscalar Processors; Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture MICRO-29, Dec. 2-4, 1996, Paris France.

Patterson, et al.; Informed Prefetching and Caching; Proceedings of the 15th ACM Symposium on Operating Systems Principles, Dec. 3-6, 1995, Copper Mountain Resort, Colorado.

Tomkins et al; Informed Multi Process Prefetching and Caching; Performance Evaluation Review Special Issue, vol. 25, No. 1, Jun. 1997—1997 ACM Sigmetrics International Conference on Measurement and Modeling of Computer Systems.

Patterson and Gibson; Exposing I/O Concurrency with Informed Prefetching; Proceedings of the Third International Conference on Parallel and Distributed Information Systems, Sep.28-30, 1994, Austin, Texas.

Shih, et al.; A File-Based Adaptive Prefetch Caching Design; Proceedings, 1990 IEEE International Conference on Computer Design; VLSI in Computers and Processors, Sep. 17-19, 1990, Hyatt Regency Cambridge, Cambridge, MA.

King, et al.; Management of a Remote Backup Copy for Disaster Recovery; ACM Transactions on Database Systems, vol. 16, No. 2, Jun. 1991, pp. 338-368.

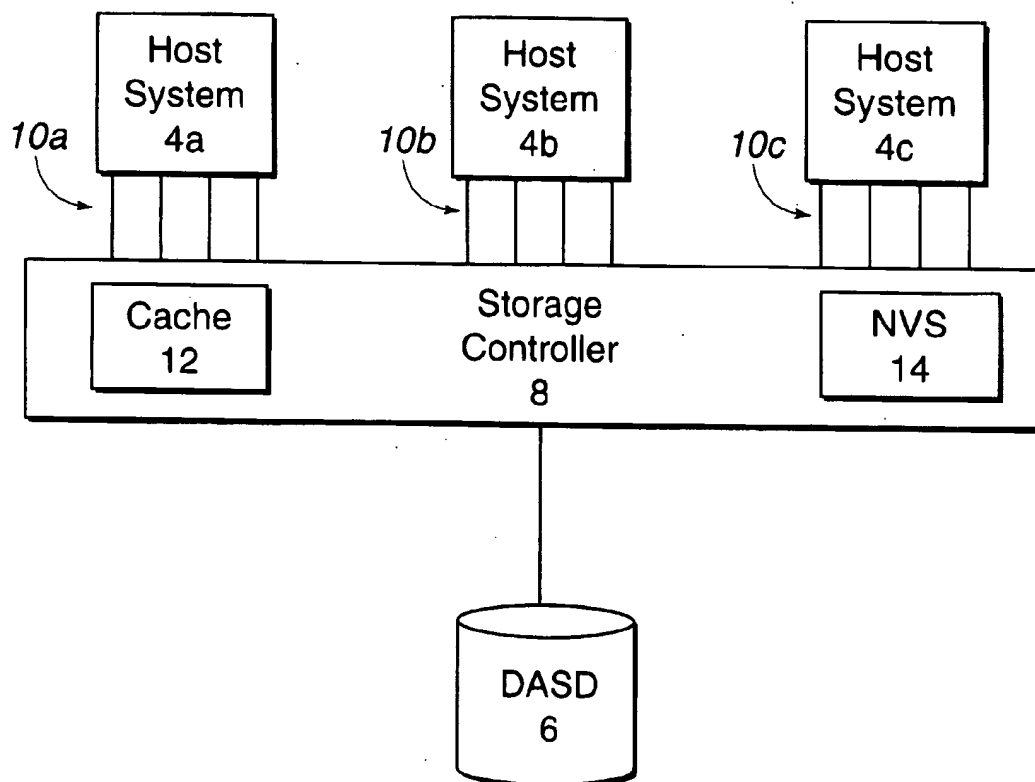
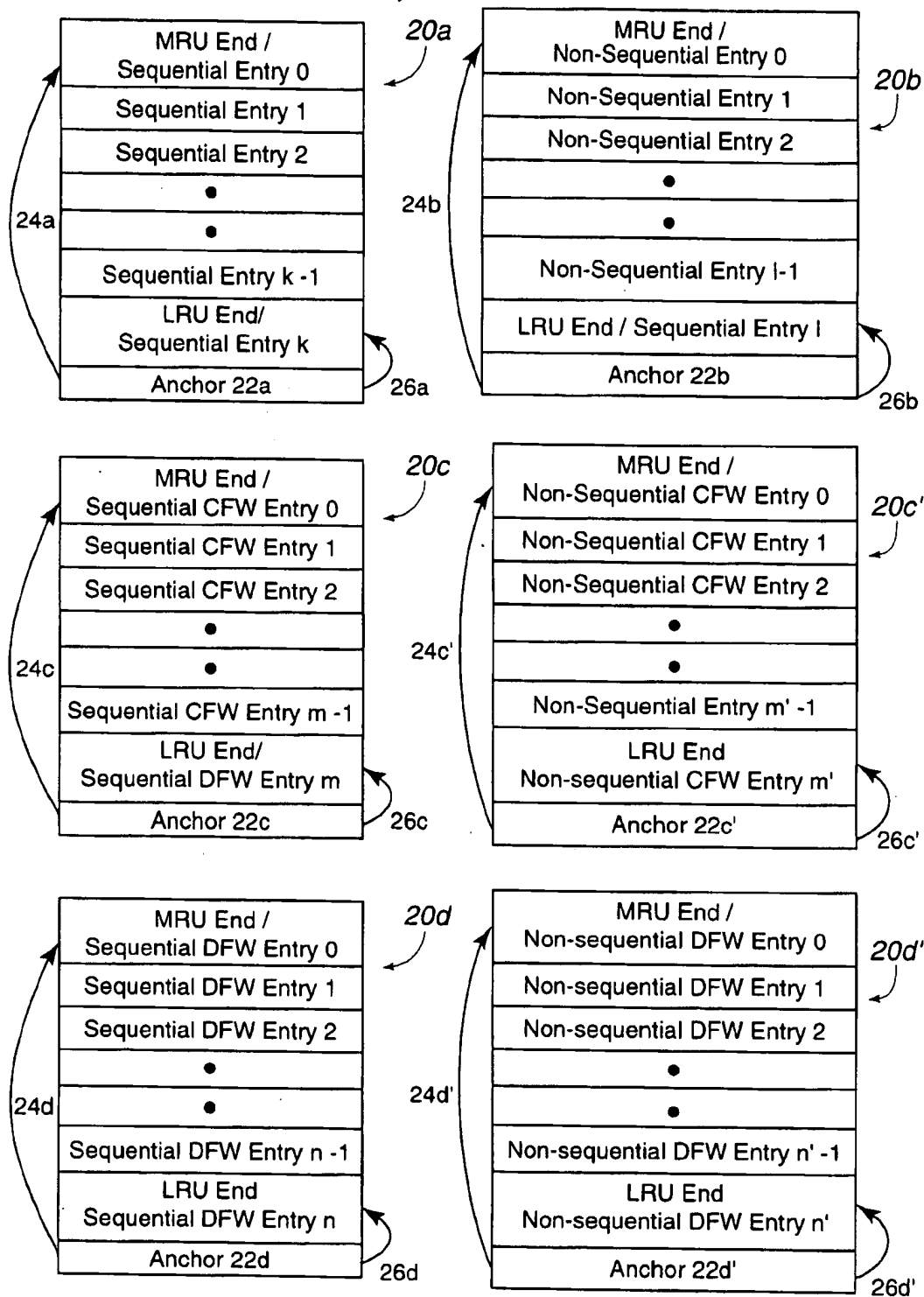
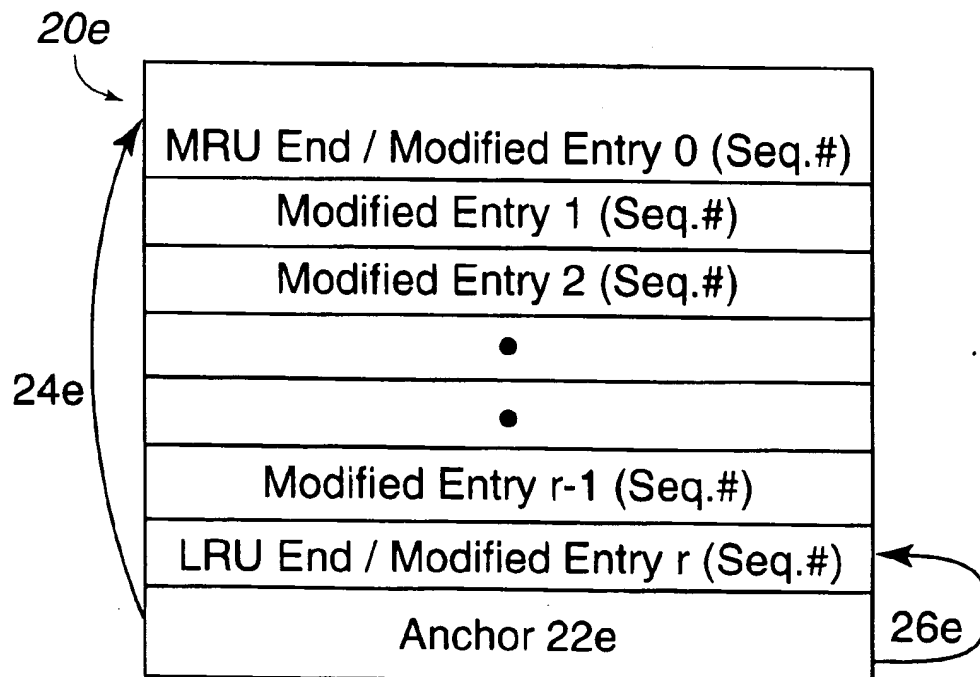
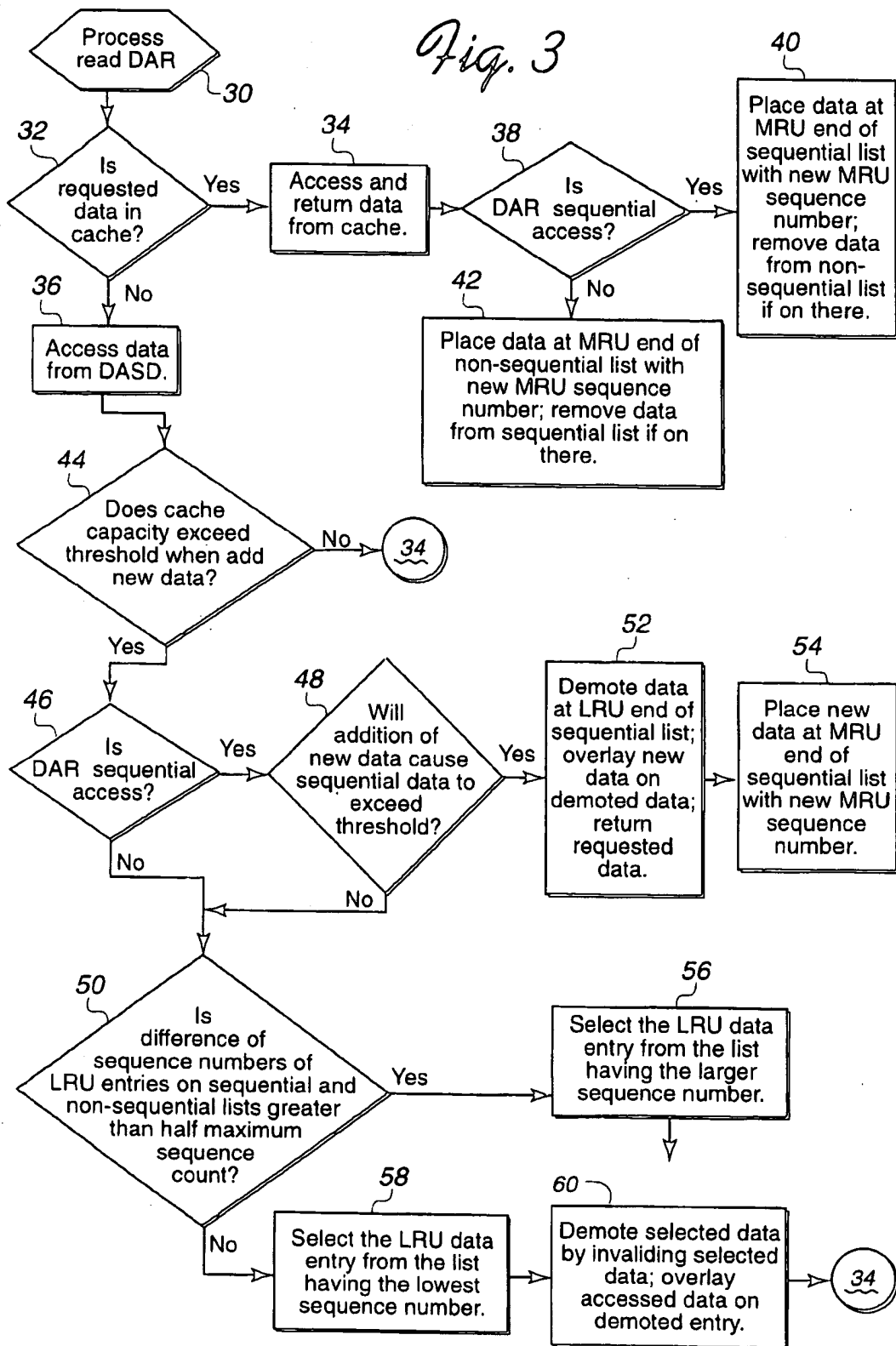
Fig. 1

Fig. 2a

*Fig. 2b*



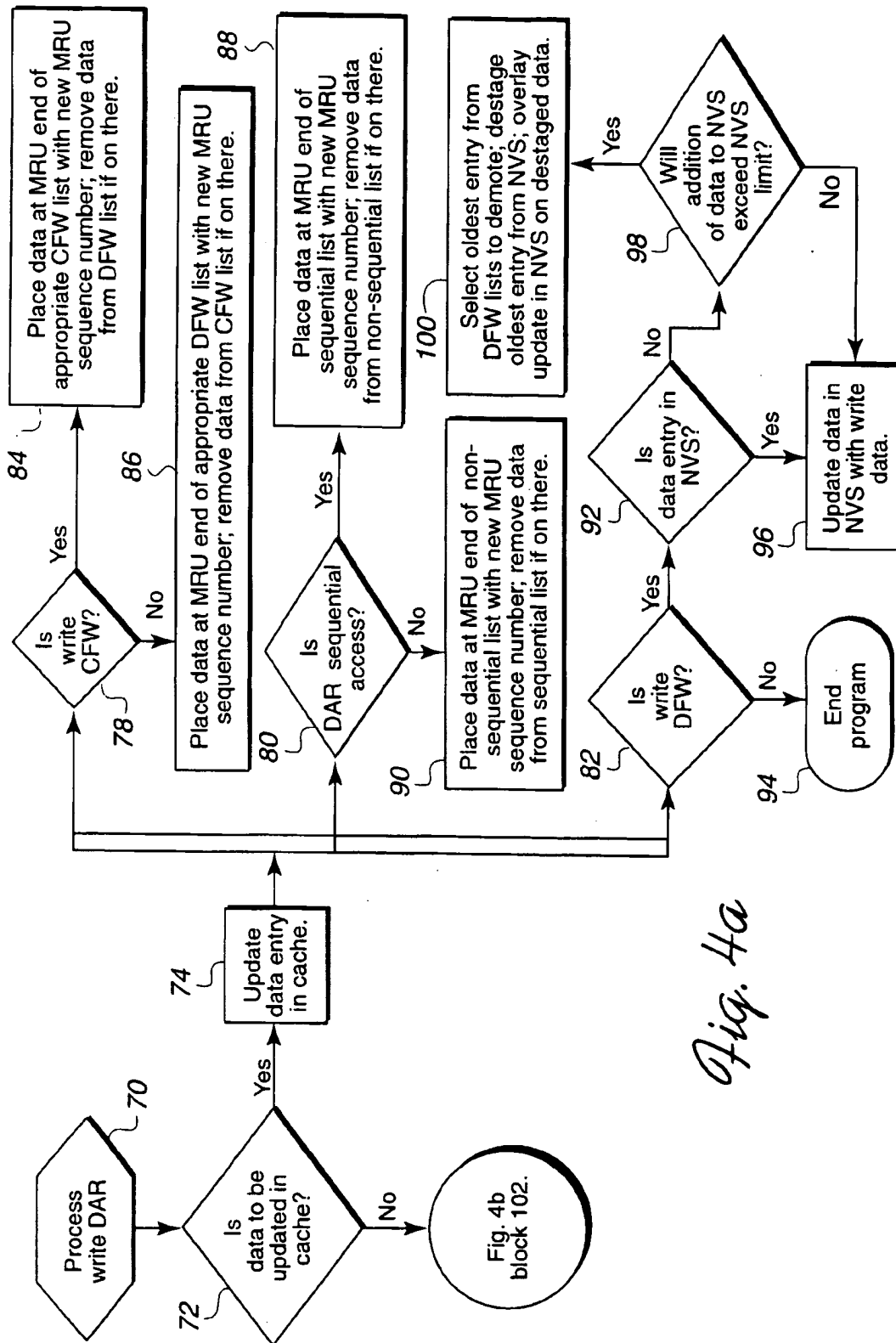
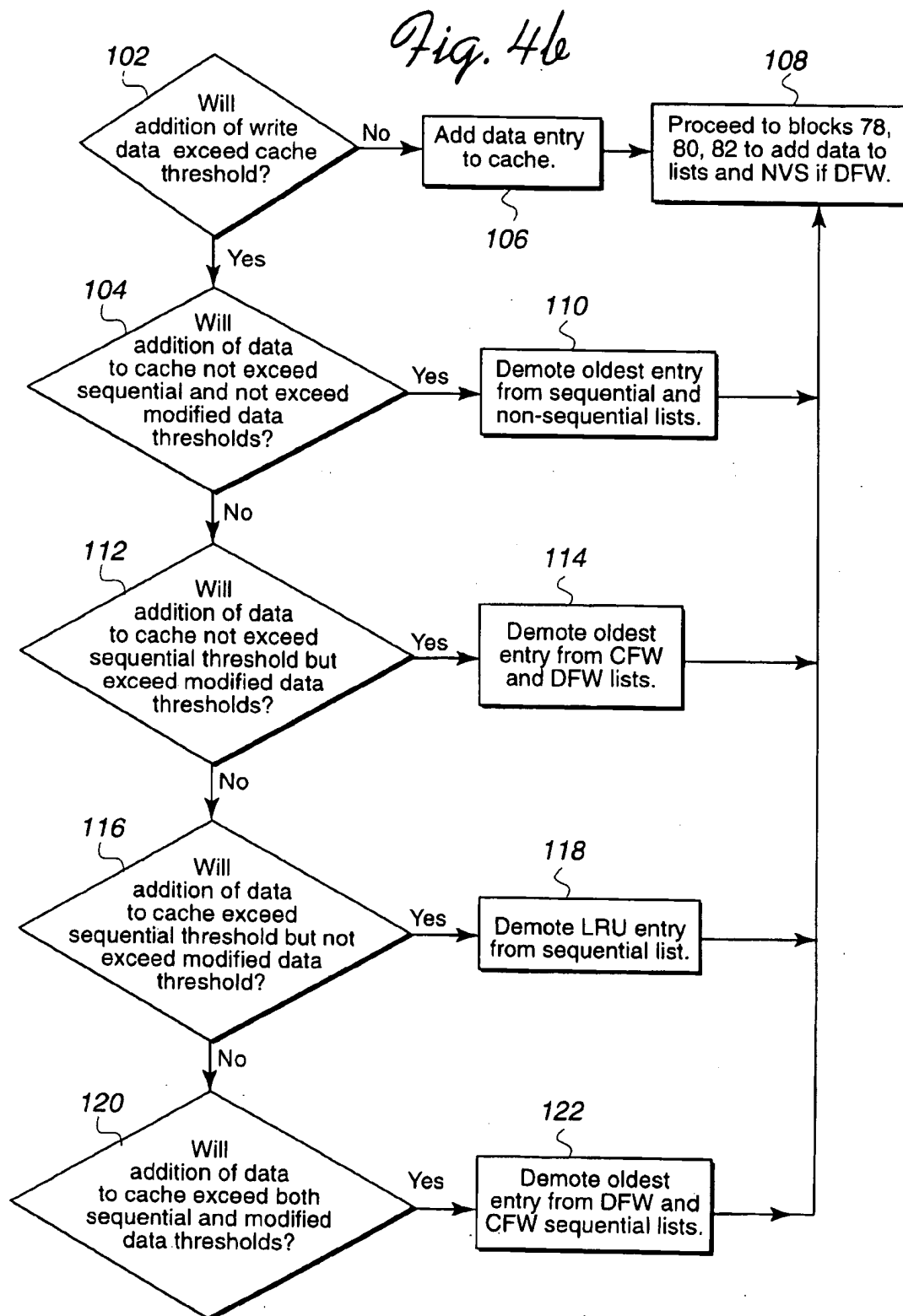
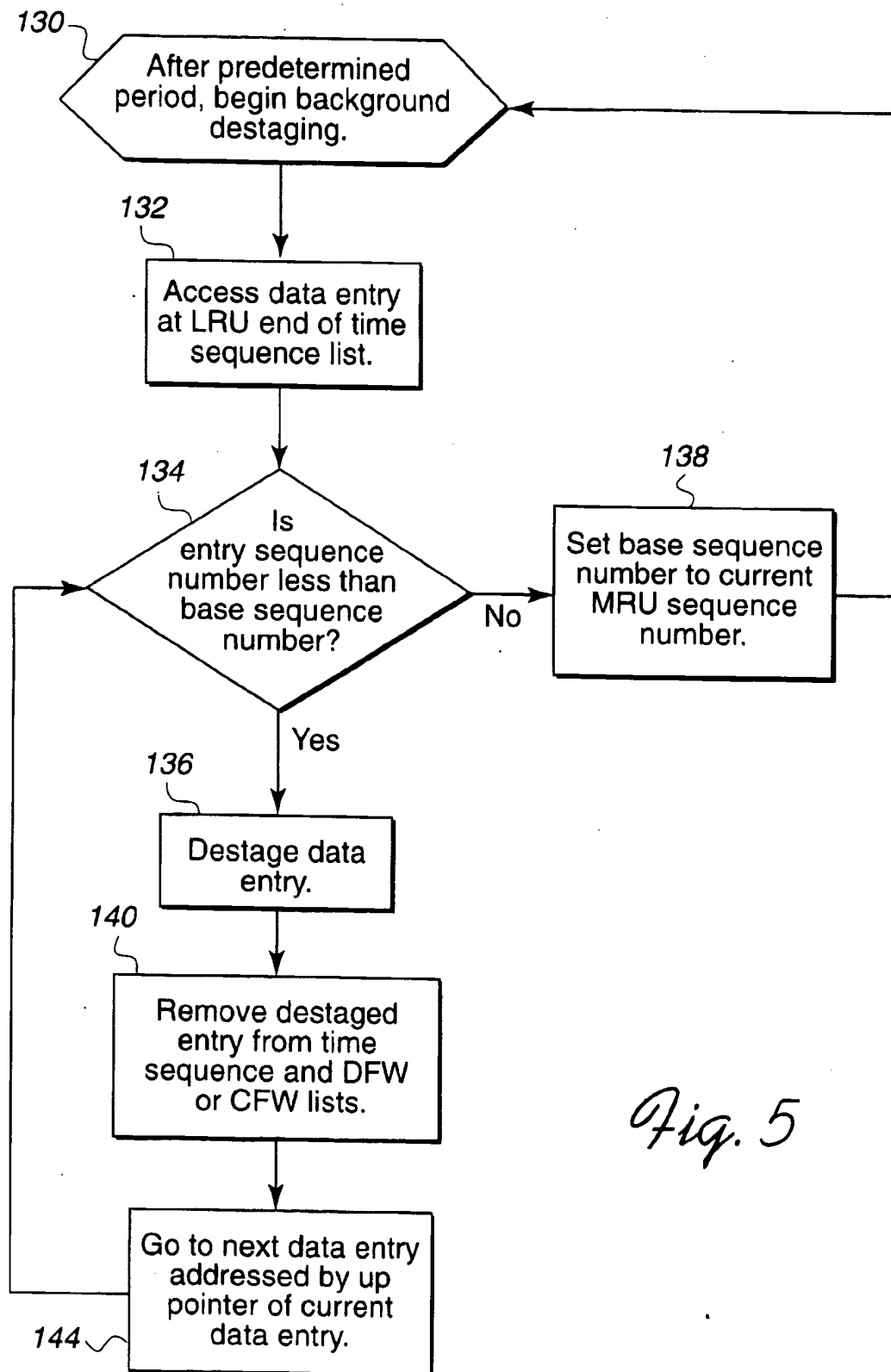


Fig. 4a





METHOD AND SYSTEM FOR MANAGING DATA IN CACHE USING MULTIPLE DATA STRUCTURES

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to a method and system for caching data and, in particular, for using multiple data structures to manage data stored in cache.

2. Description of the Related Art

Data processing systems use a high-speed managed buffer memory, otherwise known as cache, to store frequently used data that is regularly maintained in a relatively slower memory device. For instance, a cache can be a RAM that buffers frequently used data regularly stored in a hard disk drive or a direct access storage device (DASD). After a track is read from the DASD, the track will be cached in RAM and available for subsequent data access requests (DARs). In this way, a storage controller processing read requests can avoid the mechanical delays of having to physically access and read data from the DASD. Cache can also be a high speed memory to a microprocessor to store data and instructions used by the microprocessor that are regularly maintained in RAM. Processor cache would buffer data from a volatile memory device, such as a DRAM or RAM.

Often data in cache is managed according to a least recently used (LRU) replacement algorithm in which the least recently used data is demoted from the cache to make room for new data. A first-in-first-out (FIFO) algorithm may also be used. The LRU replacement algorithm works by organizing the data in the cache in a linked list of data entries which is sorted according to the length of time since the most recent reference to each data entry. The most recently used (MRU) data is at one end of the linked list, while the least recently used (LRU) data is at the other. Data that is accessed from the linked list or added for the first time is promoted to the MRU end. When data is demoted to accommodate the addition of new data, the demoted data is removed from the LRU end.

Data can be accessed sequentially or non-sequentially. In the non-sequential access mode, data records are randomly requested. Such non-sequential accesses often occur when an application needs a particular record or data sets. Sequential data access occurs when numerous adjacent tracks are accessed, such as for a data backup operation or to generate a large report. For instance, a disk backup usually creates one long sequential reference to the entire disk, thus, flooding the cache with data. One problem with LRU schemes is that if a sequential data access floods the cache when placed at the MRU end, then other non-sequential records are demoted and removed from cache to accommodate the large sequential data access. Once the non-sequential data is demoted from cache, a data access request (DAR) for the demoted data must be handled by physically accessing the data from the slower memory device.

One goal of cache management algorithms is to maintain reasonable "hit ratios" for a given cache size. A "hit" is a DAR that was returned from cache, whereas a "miss" occurs when the requested data is not in cache and must be retrieved from DASD. A "hit ratio" is empirically determined from the number of hits divided by the total number of DARs, both hits and misses. System performance is often determined by the hit ratio. A system with a low hit ratio may cause delays to application program processing while requested data is retrieved from DASD.

A low hit ratio indicates that the data often was not in cache and had to be retrieved from DASD. Low hit ratios

may occur if non-sequentially accessed data is "pushed" out of the cache to make room for a long series of sequentially accessed data. The higher probability of subsequent DARs toward non-sequentially accessed data further lowers the hit ratio because non-sequentially accessed data has a greater likelihood of being accessed. Moreover, the non-sequentially accessed data is "pushed out" of cache to make room for sequentially accessed data that has a lower likelihood of being accessed.

In certain systems, sequential data is placed at the LRU end and non-sequential data at the MRU end. Such methodologies often have the effect of providing an unreasonably low hit ratio for sequentially accessed data because the sequentially accessed data has some probability of being accessed (although usually less than non-sequentially accessed data). Algorithms that place sequentially accessed data at the LRU end cause the sequential data to be demoted very quickly, thus providing a relatively low hit ratio. Still further, if there is a continued sequence of write operations, i.e., modified data, read data could be pushed off the LRU list in cache, thus lowering the hit ratio for read accessed data.

In current storage controller systems, a battery backed up RAM or non-volatile storage unit (NVS) may maintain a shadow copy of all modified data in cache. Storage systems provided by International Business Machines Corporation ("IBM") include two write operations, a cache fast write (CFW) and a DASD fast write (DFW). In a DASD fast write operation, data is written to both the cache and the NVS unit. The DASD fast write operation allows fast write hits by maintaining two copies of all data modifications, one in cache and another in NVS storage. The non-volatile storage protects against data loss by saving the data for up to 48 hours (assuming a fully-charged battery) if power fails. When power is restored, then the data may be destaged from the NVS unit to DASD. DASD fast write applies to all write hits and to predictable writes. A write hit occurs when the requested data is in the cache.

Cache fast write (CFW) improves write operation performance for data that the user does not need to store on DASD. Because the data does not have to be stored on the DASD, cache fast write eliminates DASD access time for write hits and predictable write operations as the write data need only be stored in cache. Further, cache fast write does not use non-volatile storage. However, cache fast write data may be written to DASD during the execution of cache management algorithms. Aspects of the DASD fast write and cache fast write operations are described in IBM publication entitled "Storage Subsystem Library: IBM 3990 Storage Control Reference (Models 1, 2, and 3)", IBM document no. GA32-0099-06, (IBM Copyright 1988, 1994), which publication is incorporated herein by reference in its entirety.

When the NVS unit reaches a predetermined threshold, data in the NVS unit must be destaged from the NVS. In current systems, the entire LRU linked list including all data entries must be scanned to locate the least recently used DASD fast write data to select as a candidate for destaging from the NVS unit.

Moreover, with current systems, frequently updated data will not be destaged to disk because when the data is updated, a new time stamp is provided which places the modified data at the top of the LRU list. Thus, frequently modified data may be more susceptible to loss as a result of system failures because such data tends not to be destaged and is instead systematically placed at the MRU end of the linked list.

SUMMARY OF THE PREFERRED EMBODIMENTS

To overcome the limitations in the prior art described above, preferred embodiments disclose a cache management scheme. A first and second data structures indicate data entries in a cache. Each data structure has a most recently used (MRU) entry, a least recently used (LRU) entry, and a time value associated with each data entry indicating a time the data entry was indicated as added to the MRU entry of the data structure. A processing unit receives a new data entry. In response, the processing unit processes the first and second data structures to determine a LRU data entry in each data structure and selects from the determined LRU data entries the LRU data entry that is the least recently used. The processing unit then demotes the selected LRU data entry from the cache and data structure including the selected data entry. The processing unit adds the new data entry to the cache and indicates the new data entry as located at the MRU entry of one of the first and second data structures.

In preferred embodiments, the data structures are linked lists, wherein the MRU entry is at one end of the linked list, and the LRU entry is at the other end.

In further embodiments, the first data structure is a list of data in cache sequentially accessed and the second data structure is a list of data in cache non-sequentially accessed. In yet further embodiments, the first data structure is a list of data written as part of a first type of write operation and the second data structure is a list of data written as part of a second type of write operation. Data written to the cache as part of the first type of write operation is also written to a storage unit.

In still additional embodiments, a third data structure of data entries from the first and second data structures has data entries having a modified time value that indicates when the data entry was first modified in cache. The processing unit provides a base time value indicating a previous time value. For those data entries in the third data structure having a time value that is older than the base time value, the processing unit destages the copy of the modified data maintained in cache to the storage unit.

Preferred embodiments provide a method and system for managing data in cache in different LRU linked lists. Using separate lists, the cache can be more efficiently managed to properly select data entries for demoting. When the linked lists distinguish between non-sequentially and sequentially accessed data, sequentially accessed data may be readily accessed from the sequentially accessed linked list for demotion to prevent sequentially accessed data from dominating cache. Moreover, if separate lists are used to distinguish between a cache fast write (CFW) and DASD fast write (DFW) where data is backed up in non-volatile storage, then when data is to be destaged from the non-volatile storage, a value to destage can be readily located from the DFW list.

BRIEF DESCRIPTION OF THE DRAWINGS

Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

FIG. 1 is a block diagram illustrating a software and hardware environment in which preferred embodiments of the present invention are implemented;

FIGS. 2a, b illustrate linked lists of cache entries in accordance with preferred embodiments of the present invention;

FIG. 3 illustrates logic to process a data access request (DAR) and demote data from cache in accordance with preferred embodiments of the present invention;

FIGS. 4a, b illustrate logic to process a write request and destage data from a non-volatile storage (NVS) unit in accordance with preferred embodiments of the present invention; and

FIG. 5 illustrates logic to destage data from the NVS unit in accordance with preferred embodiments of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

In the following description, reference is made to the accompanying drawings which form a part hereof, and which is shown, by way of illustration, several embodiments of the present invention. It is understood that other embodiments may be utilized and structural changes may be made without departing from the scope of the present invention.

Hardware and Software Environment

FIG. 1 illustrates a hardware environment in which preferred embodiments are implemented. A plurality of host systems 4a, b, c are in data communication with a DASD 6 via a storage controller 8. The host systems 4a, b, c may be any host system known in the art, such as a mainframe computer, workstations, etc., including an operating system such as WINDOWS®, AIX®, UNIX®, MVS™, etc. AIX is a registered trademark of IBM; MVS is a trademark of IBM; WINDOWS is a registered trademark of Microsoft Corporation; and UNIX is a registered trademark licensed by the X/Open Company LTD. A plurality of channel paths 10a, b, c in the host systems 4a, b, c provide communication paths to the storage controller 8. The storage controller 8 and host systems 4a, b, c may communicate via any network or communication system known in the art, such as LAN, TCP/IP, ESCON®, SAN, SNA, Fibre Channel, SCSI, etc. ESCON is a registered trademark of International Business Machines Corporation ("IBM"). The host system 4a, b, c executes commands and receives returned data along a selected channel 10a, b, c. The storage controller 8 issues commands to physically position the electromechanical devices to read the DASD 6. In preferred embodiments, the structure of the storage controller 8 and interface between the storage controller 8 and host system may include aspects of the storage controller architecture described in the following U.S. patent applications assigned to IBM: "Failover System for a Multiprocessor Storage Controller," by Brent C. Beardsley, Matthew J. Kalos, Ronald R. Knowlden, Ser. No. 09/026,622, filed on Feb. 20, 1998; and "Failover and Failback System for a Direct Access Storage Device," by Brent C. Beardsley and Michael T. Benhase, Ser. No. 08/988,887, filed on Dec. 11, 1997, both of which applications are incorporated herein by reference in their entirety.

The storage controller 8 further includes a cache 12. In alternative embodiments, the cache 12 may be implemented in alternative storage areas accessible to the storage controller 8. In preferred embodiments, the cache 12 is implemented in a high speed, volatile storage area within the storage controller 8, such as a DRAM, RAM, etc. The length of time since the last use of a record in cache is maintained to determine the frequency of use of cache. Data can be transferred between the channels 10a, b, c and the cache 12, between the channels 10a, b, c and the DASD 6, and between the DASD 6 and the cache 12. In alternative embodiments with branching, data retrieved from the DASD 6 in response to a read miss can be concurrently transferred to both the channel 10a, b, c and the cache 12 and a data write can be concurrently transferred from the channel 10a, b, c to both a non-volatile storage unit and cache 12.

Also included in the storage controller 8 is a non-volatile storage (NVS) unit 14, which in preferred embodiments is a battery backed-up RAM, that stores a copy of modified data maintained in the cache 12. In this way if failure occurs and the modified data in cache 12 is lost, then the modified data may be recovered from the NVS unit 14.

To determine whether a DAR is sequentially or non-sequentially accessed, a command may be used to inform the storage controller 8 that a following DAR request is part of a series of sequentially accessed data. For instance, in the IBM mainframe environment, a Define Extent command indicates whether the following I/O operations are part of a sequential access. A description of the Define Extent commands is provided in the IBM publication, "IBM 3990/9390 Storage Control Reference," IBM Document no. GA32-0274-04 (Copyright IBM, 1994, 1996), which publication is incorporated herein by reference in its entirety. Alternatively, the storage controller 8 may utilize a predictive buffer memory management scheme to detect whether a sequence of data transfer operations are part of a sequential data access request. Such predictive buffer memory management schemes are described in U.S. Pat. No. 5,623,608, entitled "Method and Apparatus for Adaptive Circular Predictive Buffer Management," assigned to IBM, and which patent is incorporated herein by reference in its entirety. These predictive memory buffer schemes may be used to detect sequential access for SCSI or mainframe type requests when the DARs do not specifically indicate whether the DAR is part of an ongoing sequential access.

Multiple Linked Lists in Cache

FIGS. 2a, b illustrate preferred embodiments of doubly linked list data structures 20a, b, c, c', d, d', e maintained in cache 12. Each linked list 20a, b, c, c', d, d', e is comprised of a node including a list of pointers to data or track entries in the cache 12. Thus, a data entry in cache may be on multiple lists. In preferred embodiments, list 20a comprises pointers to all k sequentially accessed data entries in cache 12; list 20b comprises pointers to all l non-sequentially accessed entries in cache 12; list 20c comprises all m sequentially accessed cache fast write (CFW) entries in cache 12; list 20c' comprises all m' non-sequentially accessed cache fast write (CFW) entries in cache 12; list 20d comprises all n sequentially accessed DASD fast write (DFW) entries in cache 12; list 20d' comprises all n' non-sequentially accessed n' DASD fast write (DFW) entries in cache 12; and list 20e in FIG. 2b comprises a set of modified entries in cache 12 and a sequence number indicating when the entry was first modified in cache 12. Thus, lists 20a and 20b combined include all entries in cache and lists 20c, c' and 20d, d' combined include all modified data entries in cache.

Each list includes an anchor entry 22a, b, c, c', d, d', e which includes a pointer to the top of the list or most recently used (MRU) end 24a, b, c, c', d, d', e and a pointer to the bottom of the list or the least recently used (LRU) end 26a, b, c, c', d, d', e. As the list is doubly linked, each entry in the lists includes a pointer to the entry above, i.e., closer to the MRU end 24a, b, c, c', d, d', e referred to herein as an "up pointer" and a pointer to the entry below, i.e., closer to the LRU end 26a, b, c, c', d, d', e referred to herein as a "down pointer."

Associated with each entry in the list is an MRU sequence counter indicating a time stamp when the data was added to the linked list 20a, b, c, c', d, d', e. In preferred embodiments, when a data entry is added to the linked lists 20a, b, c, c', d,

d', e, the previous MRU sequence counter is incremented to determine the current MRU sequence number. In this way, entries with a lower sequence counter number have remained in the list unaccessed for a longer period of time, i.e., less recently used. In preferred embodiments, after the storage controller 8 accesses data, the data is added back to the MRU end 24a, b, c, c', d, d'. However, list 20e does not alter the sequence number of data entries after the data is placed in cache 12 as list 20e maintains the sequence number for when the entry was first modified in cache 12 and does not change as a result of subsequent modifications or accesses to the data entries. As entries are added at the MRU end 24a, b, c, c', d, d', e other entries move down the doubly linked list toward the LRU end 26a, b, c, c', d, d', e.

The lists 20a, b, c, c', d, d', e may be implemented as a doubly linked list of pointers to the data in cache 12 as shown in FIG. 2. Alternatively, the lists 20a, b, c, c', d, d', e may be implemented in control blocks allocated in cache 12, wherein each track or data set has a corresponding control block in the control block section of cache 12. If a track or data set was in the list 20a, b, c, c', d, d', e, then the control block for such track would include fields indicating for the data entry in cache 12 the "up pointer", "down pointer," MRU sequence number for each list 20a, b, c, c', d, d', e including the data entry. Another control block could maintain the anchor 22a, b, c, c', d, d', e information to access the beginning and end of the list. Embodiments utilizing the control blocks may be used to avoid dynamic memory allocation. In still further embodiments, the list 20a, b, c, c', d, d', e may be comprised of the actual data instead of just pointers to such data.

By providing separate lists, the storage controller 8 may more effectively manage cache 12 and determine which entries to demote or destage from the NVS unit 14. For instance, to prevent sequentially accessed data from dominating cache 12, the storage controller 8 can use the sequentially accessed list 20a to select data to demote from cache 12. Moreover, the storage controller 8 may modify the sequence number of sequentially accessed data to provide sequentially accessed data with a relatively lower sequence number than the sequence number provided non-sequentially accessed data. In this way, the storage controller 8 would be more likely to demote sequentially accessed data over non-sequentially accessed data. For instance, the storage controller 8 may accelerate or modify sequence numbers added to the sequential CFW 20c and DFW 20d lists to provide sequential modified data with a relatively lower sequence number so the sequential modified data is demoted before non-sequential modified data.

The storage controller 8 could also use the CFW 20c, c' and DFW 20d, d' lists comprising modified data to prevent modified data entries from dominating cache by selecting modified entries from the CFW 20c, c' and DFW 20d, d' lists to demote after modified data, i.e., the total of entries in the CFW 20c, c' and DFW 20d, d' lists, reaches a predetermined threshold level of cache 12.

Data in cache 12 may be on more than one list. For instance, a data track can be on one of the sequential list 20a or non-sequential list 20b, but not both, and on one of the CFW list 20c, c' or DFW list 20d, d', but not more than one of the lists, and on the modified list 20e indicating when the data was first modified in cache 12. Moreover, the sequence number for a data entry maintained on two lists may be different. For instance, a data entry on the DFW list 20d, d' has a sequence number indicating when the entry was last DASD fast write modified. If this entry is subsequently accessed sequentially as part of a read request, then this

entry would have a more current sequence number on the sequential list 20a than the DFW lists 20d, d', which remains unchanged as a result of the access. The sequential 20a and non-sequential 20b lists are updated on either a read or write data access request. Moreover, the same data entry may have an even earlier sequence number on the list 20e indicating when the data entry was first modified in cache 12.

FIG. 2b illustrates a preferred embodiment of the time sequence destage list 20e which is used to determine when to destage entries from cache 12 that were modified. At predetermined time intervals, the storage controller 8 would access the list 20e and destage all entries having a sequence number less than a certain previous time value. In this way, the storage controller 8 can insure that data will not remain unmodified and not destaged to DASD 6 for an extended period of time. Once a data entry is destaged using the time sequence list 20e, the DFW list 20d, d' or CFW list 20c, c', the data entry is removed from the time sequence list 20e.

To add a data entry to the MRU end of a list 20a, b, c, c', d, d', e, the storage controller 8 would modify the MRU pointer to point to the added data entry, adjust the up pointer of the previous MRU entry to address the new (added data entry) MRU data entry and adjust the down pointer of the new MRU entry to address the previous MRU data entry. A sequence number is maintained to indicate when a data entry was placed at the MRU end 24a, b, c, c', d, d', e of the list 20a, b, c, c', d, d', e. When placing a data entry at the MRU end 24a, b, c, c', d, d', e the storage controller 8 would increment the sequence number. To place a data entry at the MRU end 24a, b, c, c', d, d' that is already in the list, the storage controller 8 would have to first remove the data entry from the list before placing the data entry at the MRU end. To remove a data entry from a list, the up and down pointers of the removed data entry are set to null and the up and down pointers of the entries in the list 20a, b, c, c', d, d', e that addressed the removed data entry are modified to address each other. If a data entry was removed from the LRU 26a, b, c, c', d, d', e or the MRU 24a, b, c, c', d, d', e end, then the MRU 24a, b, c, c', d, d', e and LRU pointers 26a, b, c, c', d, d', e would have to be modified.

To demote a data entry from the LRU end 26a, b, c, c', d, d', e of the list 20a, b, c, c', d, d', e, the storage controller 8 would modify the LRU pointer 26a, b, c, c', d, d', e in the anchor entry 22a, b, c, c', d, d', e to address the data entry addressed by the up pointer of the entry to be demoted, set the down pointer of the data entry addressed by the new LRU data entry to null, and modify the up pointer of the data entry to demote to null.

Cache Management

FIG. 3 illustrates logic executed by the storage controller 8 to process a read request. Because a read request does not concern modified data, only the sequential 20a and non-sequential 20b lists are used. Control begins at block 30 which represents the storage controller 8 processing a read DAR. Control transfers to block 32 which represents the storage controller 8 determining whether the requested data is in cache 12. If so, control transfers to block 34; otherwise control transfers to block 36. At block 34, the storage controller 8 accesses the requested data from cache 12 and returns the accessed data to the requestor, i.e., application or device initiating the read DAR. Control then transfers to block 38 which represents the storage controller 8 determining whether the accessed read DAR is a sequential access request. If so, control transfers to block 40 which represents the storage controller 8 placing the accessed data at the

MRU end 24a of the sequential list 20a and provide the data entry the current MRU sequence number. The current MRU sequence number may be calculated by incrementing the previous sequence number. If the data entry was on the non-sequential list 20b, then the storage controller 8 would remove the data from the non-sequential list. Otherwise, if the DAR read request is non-sequential, then control transfers to block 42 which represents the storage controller placing the accessed data at the MRU end 24b of the non-sequential list 20b and, if necessary, removing the data entry from the sequential list 20a.

If the requested data is not in the cache 12, then control transfers to block 36 which represents the storage controller 8 accessing the requested data from DASD 6. Control then transfers to block 44 which represents the storage controller 8 determining whether the addition of the accessed data will exceed a cache 12 capacity threshold. If no, control transfers to block 34 et seq. to return the requested data to the requestor and add the accessed data to the MRU end 24a, b of the appropriate list 20a, b. If the cache 12 capacity threshold would be exceeded, then the storage controller 8 must demote a data entry from the LRU end 26a, b of one of the sequential or non-sequential linked lists 20a, b in cache 12.

Control transfers to block 46 which represents the storage controller determining whether the requested data is from a sequential DAR. If so, control transfers to block 48; otherwise, control transfers to block 50. Block 48 represents the storage controller 8 determining whether the addition of new data will cause the amount of sequentially accessed data in cache 12 to exceed some predetermined threshold. The storage controller 8 may make such determination by considering the total number of sequentially accessed data entries in the sequential list 20a. If the addition of the new sequentially accessed data entry will cause the limit to be exceeded, then control transfers to block 52; otherwise, control transfers to block 50. Block 52 represents the storage controller 8 demoting data at the LRU end 26a of the sequential list 20a, overlaying the new data onto the location of the demoted data, and returning the requested data. Control then transfers to block 54 where the storage controller 8 places the accessed data at the MRU end 24a of the sequential list 20a with the new MRU sequence number.

If sequential data does not have to be demoted, then the storage controller 8 must determine the sequentially or non-sequentially accessed data entry that has been in cache 12 the longest without being accessed, i.e., the LRU entry 26a, b, across both lists 20a, b. At block 50, the storage controller 8 determines whether either of the lowest sequence numbers in the lists 20a, b have wrapped. For instance, if the maximum sequence counter is 100, then any sequence number less than 100 is either that number under 100 or greater than 100 if the counter "wrapped." For instance a sequence number of 4 can be either 4, 104, 204, etc. To determine whether a wrap occurred, in preferred embodiments, the storage controller 8 may determine whether the difference of the LRU 26a, b, i.e., lowest sequence numbers from both lists 20a, b, is greater than half the maximum sequence count. For instance, if the maximum sequence count is 100 and the LRU (e.g., lowest sequence numbers) on the lists 20a, b are 3 and 94, then the difference, 91, is greater than half the maximum sequence count, 50. In such case, the storage controller 8 selects the greatest sequence number, 94, as it indicates an older sequence number than the wrapped sequence number 3.

If one of the sequence numbers wrapped, then control transfers to block 56, which represents the storage controller

8 selecting the LRU data entry 26a, b from the lists 20a, b having the larger sequence number as the candidate for demotion instead of the LRU data entry 26a, b having the lowest sequence number. Otherwise, control transfers to block 58 which represents the storage controller 8 selecting the LRU data entry 26a, b from the list 20a, b having the lowest sequence number, in which case there was not a wrap. From block 56 or 58, control transfers to block 60 which represents the storage controller 8 demoting the selected data entry by invalidating the selected data entry in cache 12 and overlaying the invalidated data entry with the new data entry. Control then transfers to block 34 to return the accessed data and place the accessed data at the MRU end 24a, b of the appropriate list 20a, b.

FIGS. 4a, b illustrate logic executed by the storage controller 8 to process a write operation to update data in the DASD 6. Control begins at block 70 which represents the storage controller 8 processing a write operation including modified data to update in the DASD 6. Control transfers to block 72 which is a decision block representing the storage controller 8 determining whether the data entry to update is already in cache 12. If so, control transfers to block 74; otherwise, control transfers to block 102 in FIG. 4b. If the data entry to update is in cache 12, then at block 74 the storage controller 8 updates the data entry in cache 12 and proceeds in parallel to execute blocks 78, 80, and 82 to place the updated data entries on the appropriate lists 20a, b, c, c', d'. At block 78, the storage controller 8 determines whether the write is a cache fast write (CFW). If the write is a CFW, then control transfers to block 84 which represents the storage controller 8 placing the updated data entry at the MRU end 24c, c' of the appropriate CFW list 20c, c' with the new MRU sequence number. Sequentially and non-sequentially written CFW data is placed on lists 20c, c', respectively. If the data entry was previously on one of the DASD fast write (DFW) lists 20d, d', then the storage controller 8 would remove the updated data entry from the DFW list 20d, d'. If the write is a DFW, then control transfers to block 86 which represents the storage controller 8 placing the updated data entry at the MRU end 24d, d' of the appropriate DFW list 20d, d' with the new sequence number. Sequentially and non-sequentially written DFW data is placed on lists 20d, d', respectively. If the data entry was previously on one of the CFW lists 20c, c', then the storage controller 8 would remove the updated data entry from the CFW list 20c, c'.

At block 80, the storage controller 8 determines whether the DAR write was a part of a series of sequential write operations. If so, control transfers to block 88 which represents the storage controller 8 placing the updated data entry at the MRU end 24a of the sequential list 20a with the new sequence number. If the data entry was previously on the non-sequential list 20b, then the storage controller 8 would remove the updated data entry from the non-sequential list 20b. If the write operation was a non-sequential operation, then control transfers to block 90 which represents the storage controller 8 placing the updated data entry at the MRU end 24b of the non-sequential list 20b with the new sequence number. If the data entry was previously on the sequential list 20a, then the storage controller 8 would remove the updated data entry from the sequential list 20a.

At block 82, the storage controller determines whether the write is a DASD fast write (DFW) operation. If so, control transfers to block 92; otherwise, control transfers to block 94 to end the program as the operation is a CFW and data does not have to be placed in the NVS unit 14. If the write is a DFW, then at block 92 the storage controller determines whether

the data entry to update is already in NVS 14. If so, control transfers to block 96 which represents the storage controller 8 updating the data entry in NVS 14 with the write update. Otherwise, if the data entry is not already in NVS 14, then control transfers to block 98 which represents the storage controller 8 determining whether the addition of data to the NVS 14 will exceed a predetermined threshold. If so, control transfers to block 100 to demote data at the LRU end 26d, d' of the DFW lists 20d, d' and destage the demoted data from the NVS 14 to DASD 16. In preferred embodiments, the storage controller 8 may select the NVS 14 entry to demote that is the oldest entry on DFW lists 20d, d'. To determine the oldest entry from the lists 20d, d' to demote, the storage controller 8 may use the logic at blocks 50 et seq. in FIG. 3 to determine whether the sequence numbers "wrapped" when selecting the oldest entry from the lists 20d, d'. The storage controller 8 would then overlay the data to update on the data entry destaged in NVS 14. In this way, the demoted entry is destaged from NVS 14 and is no longer modified data. In such case, the demoted data may be maintained in cache 12 and remain on the sequential 20a or non-sequential 20b lists. If the addition of the data update to NVS 14 will not exceed a limit, then control transfers to block 96 to add the update to NVS 14.

If the data in DASD 6 to be updated is not in cache 12, then control transfers to block 102 in FIG. 4b which represents the storage controller 8 determining whether the addition of the modified data to the cache 12 would exceed a predetermined cache 12 capacity threshold. If so, control transfers to block 104; otherwise, control transfers to block 106 which represents the storage controller 8 adding the modified data to cache and then proceeding to block 108 where the storage controller executes blocks 78, 80, and 82 in parallel to modify the appropriate lists 20a, b, c, c', d, d' to reflect the data added to cache, and to the NVS 14 if the data is a DFW. If data needs to be demoted from cache 12 to make room for the new write data, then at block 104 the storage controller 8 determines whether the addition of the modified data to cache 12 will cause a sequential and modified data thresholds to be exceeded.

The sequential threshold is exceeded when the number of sequentially accessed data entries in cache exceed a predetermined threshold. The current number of sequential entries in cache can be determined by examining the sequential list 20a. The modified data threshold is exceeded when the number of modified data entries exceeds a predetermined threshold. The current number of modified data entries in cache 12 can be determined by examining the total number of entries in the CFW 20c, c' and DFW 20d, d' lists. If the addition of the modified data to cache will not exceed both the sequential and modified thresholds, then control transfers to block 110; otherwise control transfers to block 112. Block 110 represents the storage controller 8 demoting the oldest data entry at the LRU ends 26a, b of the sequential 20a and non-sequential 20b lists. The storage controller 8 may apply the logic of blocks 50 et seq. in FIG. 3 to select the oldest entry in the event there is a wrap. After demoting the oldest entry in cache 12, then control transfers to block 108 to update the lists 20a, b, c, c', d, d' by concurrently executing blocks 78, 80, and 82.

If the addition of data to cache 12 exceeds either sequential or modified thresholds or both, then at block 112, the storage controller 8 determines whether the addition of the modified data entry to cache 12 will not exceed the sequential threshold, but exceed the modified threshold. If so, control transfers to block 114; otherwise, control transfers to block 116. At block 114, the storage controller 8 demotes the

oldest entry from the CFW 20c, c' and DFW 20d, d' lists. This ensures that a modified data entry is demoted so the addition of the new modified entry does not cause the modified data threshold to be exceeded or further exceeded. The storage controller 8 may apply the logic at blocks 50 et seq. in FIG. 3. To select the two LRU entries from the lists 20c, c', d, d' to compare according to the logic at blocks 50 et seq., the storage controller 8 may select the oldest (lowest sequence number) and most recent (highest sequence number) of the LRU entries 26c, c', d, d' to compare. From block 114, control transfers to block 108 to update the appropriate lists to reflect the new modified data added to the cache 12. At block 116, the storage controller determines whether the addition of the new modified data to cache 12 will cause the sequential threshold to be exceeded, but not the modified threshold to be exceeded. If so, control transfers to block 118 which represents the storage controller 8 demoting the LRU entry 26a from the sequential 20a list to not increase the current number of sequentially accessed entries in cache 12.

Block 120 represents the storage controller determining whether the addition of the modified data to cache will cause both the sequential and modified thresholds to be exceeded. If so, control transfers to block 122 to demote the oldest LRU entry 26c, d from the CFW 20c and DFW 20d sequential lists using the logic at block 50 et seq. in FIG. 3. The operation at block 122 insures that modified sequential data is demoted from cache 12. The storage controller 8 would also modify one or more of the lists 20a, b, c, c', d, d' to reflect the removal of the demoted data entry. In this way the modified 20a, b, c, c', d, d' lists are used to ensure that data is demoted in a manner that will not exceed certain thresholds of data types in the cache 12, such as sequentially accessed data and modified data.

In further embodiments, the storage controller 8 may adjust the sequence number of sequentially accessed data downward before placing sequentially accessed data on the sequential lists 20a, c, d, thereby accelerating the advancement of sequentially accessed data as a candidate for removal from the cache 12. An example of an accelerated sequence number for sequentially accessed data could be calculated from equation (1) as follows:

$$\text{accelerated sequence number} = \text{MRU sequence count} - \frac{1}{2} * (\text{LRU list member count}) \quad (1)$$

Thus, the accelerated sequence number reduces the current MRU sequence number by one-half the number of data entries in the list to which the data entry will be added. Those skilled in the art will appreciate that alternative equations and methods could be used to adjust the sequence number for sequentially accessed data to make such data a more likely candidate for demotion than non-sequentially accessed data.

The time sequence list 20e in FIG. 2b includes data entries, i.e., pointers to data in cache 12, and a sequence number indicating when the data entry was first modified in cache 12. Thus, data may have been added to cache as part of a read request, but will not be added to the time sequence list 20e until such data has been modified in cache 12. The time sequence list may be a user specified subset of tracks in cache 12 or all modified data in cache 12 as indicated in the CFW 20c, c' and DFW 20d, d' lists. If the data entry is subsequently modified again, then the sequence number on the time sequence list 20e remains unchanged even though it may change on the CFW 20c, c' or DFW 20d, d' list. The storage controller 8 further maintains a base sequence number indicating a previous time stamp. In preferred

embodiments, the storage controller 8 may destage all data entries from the time sequence list 20e having a sequence number that is older than the base sequence number.

When modified data is added to the cache 12 or when a current data entry is updated for the first time, then the storage controller 8 would add the new modified data entry to the time sequence list 20e and indicate the sequence number when the modified data was first added. This step of adding a new modified entry to the time sequence list 20e may occur concurrently with the other parallel operations 78, 80, and 82 in FIG. 4a at the first instance the data is modified in cache. Subsequent modifications of the data entry will not change the entries in the time sequence list 20e.

FIG. 5 illustrates logic executed by the storage controller 8 to periodically destage data from the cache 12 and/or NVS 14 using the time sequence list 20e. When the storage controller 8 initiates background destaging, the storage controller 8 sets a base sequence number to the current MRU sequence number. After a predetermined period interval, the storage controller 8 again initiates the background destaging routine using the base sequence number set during the execution of the previous background destaging operation to determine if any data entries in the time sequence list 20e have sequence numbers (indicating the time the data was first modified) less than the base sequence number. Such a situation may indicate that although the data entry was placed on the list a while ago, the data entry has, nonetheless, not been destaged. The logic of FIG. 5 ensures that frequently updated data, which has its MRU sequence number frequently reset to the current time value, gets destaged to DASD 6. When a time sequence number is less than the base sequence number, then the modified data was likely updated and, in such case, may not be demoted using the other lists 20a, b, c, c', d, d'. With the logic of FIGS. 3 and 4a, b, such frequently updated (modified) data would not otherwise be destaged because the sequence number of the modified data in the lists 20a, b, c, c', d, d' is reset each time the data is accessed and placed back at the MRU end 24a, b, c, c', d, d' of the lists 20a, b, c, c', d, d' according to the logic of FIGS. 3, 4a, b.

With reference to FIG. 5, control begins at block 130 which represents the storage controller 8 periodically initiating a background destaging operation at predetermined time intervals. Control transfers to block 132 which represents the storage controller 8 accessing the data entry at the LRU end 26e of the time sequence list 20e. Control then transfers to block 134 which represents the storage controller 8 determining whether the sequence number of the accessed data entry in the list 20e is less than the base sequence number, i.e., whether the data entry sequence number is older than the base sequence number. If so, control transfers to block 136 which represents the storage controller 8 destaging the accessed data entry. To destage the modified data, the copy of the modified data in the cache 12 or NVS unit 14 is destaged to the DASD 6. Control transfers to block 140 which represents the storage controller 8 removing the destaged data from the CFW 20c, c' or DFW 20d, d' lists and the time sequence list 20e. However, the destaged data may still remain in cache 12 and on the sequential 20a and non-sequential 20b lists as unmodified data. Control then transfers to block 144 which represents the storage controller 8 accessing the next data entry in the list closer to the MRU end 24e addressed by the up pointer of the current, just considered, data entry. From block 144 control proceeds back to block 134 to process further entries on the list 20e.

If the data entry sequence number is not less than the base sequence number, then control transfers to block 138 which represents the storage controller 8 setting the base sequence number to the current MRU sequence number and then back to block 130 to initiate the background destaging operation after a predetermined interval. If the accessed data entry sequence number is greater than the base sequence number, then the data entry is more recent than the base sequence number. As the list is scanned from the LRU end 26e to the MRU end 24e, no further data entries in the time sequence list 20e need be destaged. Once the considered data entry in list 20e has a more recent time value than the base sequence number, no further entries closer to the MRU end 24e would have a time value later than the base sequence number.

Alternative Embodiments and Conclusion

The preferred embodiments may be implemented as a method, apparatus or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The term "article of manufacture" (or alternatively, "computer program product") as used herein is intended to encompass one or more computer programs and data files accessible from one or more computer-readable devices, carriers, or media, such as a magnetic storage media, "floppy disk," CD-ROM, a file server providing access to the programs via a network transmission line, holographic unit, etc. Of course, those skilled in the art will recognize that many modifications may be made to this configuration without departing from the scope of the present invention.

Preferred embodiments were described with respect to the IBM mainframe environment, where a storage controller unit interfaces numerous host systems with a DASD. However, those skilled in the art will appreciate that the preferred caching algorithms could apply to any data transfer interface known in the art, including SCSI, ST-506/ST-412, IDE/ATA, Enhanced Small Device Interface (ESDI), floppy disk, parallel port, ATA, EIDE, ATA-2, Fast ATA, Ultra ATA, etc., where data is cached.

Preferred embodiments were described with respect to a sequence number used to determine the length of time data has remained unaccessed in the cache. In alternative embodiments, different methods may be used to maintain time information for data entries in the cache, such as a time stamp.

The logic of FIGS. 3-5 may be implemented in microcode accessible to the storage controller 8 or as part of an application the storage controller 8 executes. Still further, the logic of FIGS. 3-5 may be executed in hardwired circuitry dedicated to managing the cache 12. Alternatively, certain of the logic of FIGS. 3-5 may be performed by the host system 4a, b, c. The logic of FIGS. 3-5 is for illustrative purposes. Certain steps may be modified or removed altogether and other steps added. Further, the order of the steps performed may also vary from the described embodiments.

In preferred embodiments, the data maintained in cache may be any data set or format, including fixed block CKD track, record, stripe, etc. Moreover, preferred embodiments were described as removing and adding data to cache 12 in response to DARs. In alternative embodiments, the logic managing the cache may add and remove data as part of data management operations unrelated to specific DARs.

Preferred embodiments were described using seven lists 20a, b, c, c', d, d', e together to manage cache. However, in alternative embodiments, the lists may be used separately and independently of each other. For instance, only the sequential 20a and non-sequential 20b lists may be used to prevent sequentially accessed data from dominating cache;

only the CFW 20c, c' or DFW 20d, d' lists may be used to destage data; and only the time sequence list 20e may be used to insure that modified data is destaged as of a particular time. In yet further embodiments, there may be different lists, such as a single DFW or CFW list of all DFW or CFW entries, both sequential and non-sequential.

In preferred embodiments, the lists were described as implemented as doubly linked list data structures comprised of lists or control blocks. Those skilled in the art will appreciate that alternative data structures may be utilized to implement the lists.

Preferred embodiments were described with respect to managing a cache that buffers data from a DASD. The logic of the preferred embodiments could be used to manage cache that buffers data from any type of memory device, non-volatile as well as volatile, to another cache memory, which may be of a higher speed providing faster access. For instance, data from a DRAM or RAM can be buffered in a higher speed cache, such as a cache that is on-board a microprocessor, e.g., the L2 cache used with the PENTIUM® II microprocessor. PENTIUM II is a registered trademark of Intel Corporation.

In summary, preferred embodiments disclose a cache management scheme using multiple data structures. A first and second data structures indicate data entries in a cache. Each data structure has a most recently used (MRU) entry, a least recently used (LRU) entry, and a time value associated with each data entry indicating a time the data entry was indicated as added to the MRU entry of the data structure. A processing unit receives a new data entry. In response, the processing unit processes the first and second data structures to determine a LRU data entry in each data structure and selects from the determined LRU data entries the LRU data entry that is the least recently used. The processing unit then demotes the selected LRU data entry from the cache and data structure including the selected data entry. The processing unit adds the new data entry to the cache and indicates the new data entry as located at the MRU end of one of the first and second data structures.

The foregoing description of the preferred embodiments of the invention has been presented for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the invention be limited not by this detailed description, but rather by the claims appended hereto. The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.

What is claimed is:

1. A method for caching data, comprising the steps of:

providing a first and second data structures indicating data entries in a cache, wherein each data structure has a most recently used (MRU) entry, a least recently used (LRU) entry, and a time value associated with each data entry indicating a time the data entry was indicated as added to the MRU entry of the data structure;

receiving a new data entry;

processing the first and second data structures to determine a LRU data entry in each data structure;

selecting from the determined LRU data entries the LRU data entry that is the least recently used;

demoting the selected LRU data entry from the cache and data structure including the selected data entry;

adding the new data entry to the cache; and
indicating the new data entry as located at the MRU entry
of one of the first and second data structures.

2. The method of claim 1, wherein the first data structure
indicates data entries in the cache sequentially accessed and
the second data structure indicates data entries in the cache
non-sequentially accessed, wherein the step of selecting the
LRU data entries comprises the steps of:

determining whether adding the new data entry to cache
would cause the number of sequentially accessed data
entries to exceed a threshold, wherein the step of
selecting from the determined LRU data entries the
LRU data entry that is least recently used occurs after
determining that adding the new data entry to cache
would not cause the sequentially accessed data entries
to exceed the threshold; and

selecting the LRU data entry from the first data structure
to demote after determining that adding the new data
entry to cache would cause the sequentially accessed
data entries to exceed the threshold.

3. The method of claim 2, further comprising the steps of:
receiving a data access request for requested data in the
cache;

returning the requested data from the cache;

determining whether the data access request is a sequen-
tial access;

indicating that the requested data is located at the MRU
entry of the first data structure after determining that
the data access request is a sequential access; and

indicating that the requested data is located at the MRU
entry of the second data structure after determining that
the data access request is a non-sequential access.

4. The method of claim 1, wherein the first data structure
indicates data entries written to the cache as part of a first
type of write operation and the second data structure indi-
cates data entries written to the cache as part of a second type
of write operation, wherein data written to the cache as part
of the first type of write operation is also written to a first
storage unit, and wherein the new data entry comprises data
to write to a second storage unit, further comprising the steps
of:

determining whether the new data entry is of the first write
operation type;

selecting the LRU data entry from the first data structure
to destage from the first storage unit to the second
storage unit after determining that the new data entry is
of the first write operation type;

indicating the selected LRU data entry as removed from
the first data structure; and

adding the new data entry to the first storage unit, wherein
the step of indicating the new data entry as located at
the MRU entry of one of the first and second data
structures comprises indicating the new data entry as
located at the MRU entry of the first data structure.

5. The method of claim 4, further comprising the steps of:
receiving modified data for a data entry that is already in
cache;

determining whether the received modified data is of the
first type of write operation;

updating the data entry in cache with the received modi-
fied data;

updating the data entry in the first storage unit with the
received modified data after determining that the
received modified data is of the first type of write
operation;

indicating that the updated data entry is at the MRU entry
of the first data structure after determining that the
modified data is of the first type of write operation; and
indicating that the updated data entry is at the MRU entry
of the second data structure after determining that the
data access request is of the second type of write
operation.

6. The method of claim 4, further comprising the steps of:
providing a third data structure indicating data entries in
cache sequentially accessed and a fourth data structure
indicating data entries in cache non-sequentially
accessed, wherein the third and fourth data structures
have an MRU entry and an LRU entry;

determining whether the new data entry is sequentially
accessed data;

indicating that the new data entry is at the MRU entry of
the third data structure after determining that the new
data entry is sequentially accessed; and

indicating that the new data entry is at the MRU entry of
the fourth data structure after determining that the new
data entry is non-sequentially accessed.

7. The method of claim 1, further comprising a third and
fourth data structures, wherein the first data structure indi-
cates modified data entries written in cache and a storage
unit as part of a sequential write operation, wherein the
second data structure indicates modified data entries written
in cache and the storage unit as part of a non-sequential
write operation, wherein the third data structure indicates modi-
fied data entries written in cache and not the storage unit as
part of a sequential write operation, and wherein the fourth
data structure indicates modified data entries written in
cache and not in the storage unit as part of a non-sequential
write operation, wherein the step of processing the first and
second data structures further comprises processing the third
and fourth data structures to determine an LRU data entry
from the first, second, third, and fourth data structures.

8. The method of claim 7, further comprising the step of
providing a modified data threshold indicating a maximum
number of modified data entries to maintain in cache,
wherein the step of processing the four data structures
occurs after determining that the addition of modified data to
the cache will cause the modified data threshold to be
exceeded.

9. The method of claim 8, further comprising a fifth data
structure indicating the number of entries in the cache
sequentially accessed and a sixth data structure indicating
the number of entries in cache non-sequentially accessed,
further comprising the steps of:

providing a sequentially accessed threshold indicating a
maximum number of sequentially accessed data entries
to maintain in cache;

determining whether the addition of a data entry to the
cache will cause the number of data entries in cache to
exceed the modified and sequentially accessed
thresholds, wherein the step of processing the first,
second, third, and fourth data structures occurs after
determining that the addition of modified data to the
cache will exceed the modified data threshold and not
exceed the sequentially accessed threshold;

processing the fifth and sixth data structures to determine
LRU data entries after determining that the modified
threshold is not exceeded, wherein the steps of select-
ing and demoting comprises selecting and demoting the
LRU data entry that is the least recently used entry from
the fifth and sixth data structures; and

processing the first and third data structures to determine
LRU data entries after determining that the modified

and sequentially accessed thresholds are both exceeded, wherein the steps of selecting and demoting comprises selecting and demoting the LRU data entry that is the least recently used entry from the first and third data structures.

10. The method of claim 1, wherein the time value cannot exceed a maximum time value, wherein after reaching the maximum time value the time value resets to zero, further comprising the step of determining whether the time value for at least one of the LRU data entries was reset to zero, wherein the step of selecting the LRU data entry comprises selecting the LRU data entry that has a time value that was previously reset.

11. The method of claim 10, wherein the time value comprises a sequence number that is incremented when data entries are indicated as added to the MRU entry of the data structures, wherein the maximum time value comprises a maximum sequence number, and wherein the LRU data entry in a data structure has the lowest sequence number in the data structure, wherein the step of determining whether the time value for an LRU data entry was reset to zero comprises the steps of:

(i) determining a difference between the sequence numbers of the LRU data entries in the first and second data structures; and

(ii) determining whether the difference between the sequence numbers is greater than half the maximum sequence number; and

wherein the step of selecting the LRU data entry from the data structures comprises the steps of:

(i) selecting the LRU data entry having a largest sequence number after determining that the difference between the LRU data entries is greater than half the maximum sequence number; and

(ii) selecting the LRU data entry having a lowest sequence number after determining that the difference between the LRU data entries is less than half the maximum sequence number.

12. The method of claim 1, further comprising a third data structure indicating data entries in the first and second data structures, wherein each data entry indicated in the third data structure has a modified time value indicating when the data entry was first modified in cache, further comprising the steps of:

executing a routine at predetermined intervals to destage data from cache to a storage unit;

providing a base time value indicating a previous time value;

determining, when executing the routine, data entries in the third data structure having a time value that is older than the base time value;

destaging, for each determined data entry in the third data structure, the copy of the modified data maintained in cache to the storage unit; and

indicating the destaged data entry as removed from the third linked list.

13. A system for caching data, comprising:

a processing unit;

a cache including data entries;

a memory area including first and second data structures accessible to the processing unit, wherein the data structures indicate data entries in the cache, wherein each data structure has a most recently used (MRU) entry, a least recently used (LRU) entry, and a time value associated with each data entry indicating a time

the data entry was indicated as added to the MRU entry of the data structure;

program logic executed by the processing unit, comprising:

(i) means for processing the first and second data structures to determine a LRU data entry in each data structure;

(ii) means for selecting from the determined LRU data entries the LRU data entry that is the least recently used;

(iii) means for demoting the selected LRU data entry from the cache and data structure including the selected data entry;

(iv) means for adding a new data entry to the cache; and

(v) means for indicating the new data entry as located at the MRU entry of one of the first and second data structure.

14. The system of claim 13, wherein the data structures are linked lists, wherein the MRU entry is at one end of the linked list and the LRU entry is at another end of the linked list.

15. The system of claim 13, wherein the cache and memory area are included in a first memory device, further comprising a second memory device, wherein the cache stores data accessed from the secondary memory device.

16. The system of claim 13, wherein the first data structure indicates data entries in the cache sequentially accessed and the second data structure indicates data entries in the cache non-sequentially accessed, wherein the program logic further comprises:

means for determining whether adding the new data entry to cache would cause the number of sequentially accessed data entries to exceed a threshold; and

means for selecting the LRU data entry from the first data structure to demote after determining that adding the new data entry to cache would cause the sequentially accessed data entries to exceed the threshold.

17. The system of claim 16, wherein the program logic further comprises:

means for receiving a data access request for requested data in the cache;

means for returning the requested data from the cache;

means for determining whether the data access request is a sequential access;

means for indicating that the requested data is located at the MRU entry of the first data structure after determining that the data access request is a sequential access; and

means for indicating that the requested data is located at the MRU entry of the second data structure after determining that the data access request is a non-sequential access.

18. The system of claim 13, wherein the first data structure indicates data entries written to the cache as part of a first type of write operation and the second data structure indicates data entries written to the cache as part of a second type of write operation, further comprising:

a first storage unit, wherein data written to the cache as part of the first type of write operation is also written to a first storage unit;

a second storage unit;

wherein the program logic further comprises:

(i) means for determining whether the new data entry is part of the first write operation;

(ii) means for selecting the LRU data entry from the first data structure to destage from the first storage

unit to the second storage unit after determining that the new data entry is part of the first write operation; and

(iii) means for adding the new data entry to the first storage unit, wherein the step of indicating the new data entry as located at the MRU entry of one of the first and second data structure comprises indicating the new data entry as located at the MRU entry of the first data structure.

19. The system of claim 18, wherein the program logic further comprises:

means for processing modified data for a data entry that is already in cache;

means for determining whether the modified data is one of the first type of write operation;

means for updating the data entry in cache with the received modified data;

means for updating the data entry in the first storage unit with the received modified data after determining that the received modified data is of the first type of write operation;

means for indicating that the updated data entry is at the MRU entry of the first data structure after determining that the modified data is of the first type of write operation; and

means for indicating that the updated data entry is at the MRU entry of the second data structure after determining that the data access request is of the second type of write operation.

20. The system of claim 13, wherein the memory area further comprises a third data structure indicating data entries in the first and second data structures, wherein each data entry in the third data structure has a modified time value indicating when the data entry was first modified in cache, wherein the program logic further comprises:

means for executing a routine at predetermined intervals to destage data from cache to a storage unit;

means for providing a base time value indicating a previous time value;

means for determining, when executing the routine, data entries in the third data structure having a time value that is older than the base time value; and

means for destaging, for each determined data entry in the third data structure, the copy of the modified data maintained in cache to the storage unit; and

means for indicating the destaged data entry as removed from the third data structure.

21. An article of manufacture for use in programming a processing unit to manage cache, the article of manufacture comprising at least one computer readable storage device including at least one computer program embedded therein that causes the processing unit to perform the steps:

providing a first and second data structures indicating data entries in a cache, wherein each data structure has a most recently used (MRU) entry, a least recently used (LRU) entry, and a time value associated with each data entry indicating a time the data entry was indicated as added to the MRU entry of the data structure;

receiving a new data entry;

processing the first and second data structures to determine a LRU data entry in each data structure;

selecting from the determined LRU data entries the LRU data entry that is the least recently used;

demoting the selected LRU data entry from the cache and data structure including the selected data entry;

adding the new data entry to the cache; and

indicating the new data entry as located at the MRU entry of one of the first and second data structures.

22. The article of manufacture of claim 21, wherein the first data structure indicates data entries in cache sequentially accessed and the second data structure indicates data entries in cache non-sequentially accessed, wherein the step of selecting the LRU data entries comprises causing the processing unit to perform the steps of:

determining whether adding the new data entry to cache would cause the number of sequentially accessed data entries to exceed a threshold, wherein the step of selecting from the determined LRU data entries the LRU data entry that is least recently used occurs after determining that adding the new data entry to cache would not cause the sequentially accessed data entries to exceed the threshold; and

selecting the LRU data entry from the first data structure to demote after determining that adding the new data entry to cache would cause the sequentially accessed data entries to exceed the threshold.

23. The article of manufacture of claim 22, further comprising the steps of:

receiving a data access request for requested data in the cache;

returning the requested data from the cache;

determining whether the data access request is a sequential access;

indicating that the requested data is located at the MRU entry of the first data structure after determining that the data access request is a sequential access; and

indicating that the requested data is located at the MRU entry of the second data structure after determining that the data access request is a non-sequential access.

24. The article of manufacture of claim 21, wherein the first data structure comprises data entries written to the cache as part of a first type of write operation and the second data structure comprises data entries written as part of a second type of write operation, wherein data written to the cache as part of the first type of write operation is also written to a first storage unit, and wherein the new data entry comprises data to write to a second storage unit, further comprising the steps of:

determining whether the new data entry is of the first write operation type;

selecting the LRU data entry from the first data structure to destage from the first storage unit to the second storage unit after determining that the new data entry is of the first type of write operation type;

indicating the selected LRU data entry as removed from the first data structure; and

adding the new data entry to the first storage unit, wherein the step of indicating the new data entry as located at the MRU entry of one of the first and second data structures comprises indicating the new data entry as located at the MRU entry of the first data structure.

25. The article of manufacture of claim 24, further comprising the steps of:

receiving modified data for a data entry that is already in cache;

determining whether the received modified data is of the first type of write operation;

updating the data entry in cache with the received modified data;

21

updating the data entry in the first storage unit with the received modified data after determining that the received modified data is of the first type of write operation;

indicating that the updated data entry is at the MRU entry of the first data structure after determining that the modified data is of the first type of write operation; and

indicating that the updated data entry is at the MRU entry of the second data structure after determining that the modified data is of the second type of write operation.

26. The article of manufacture of claim 25, further comprising the steps of:

providing a third data structure indicating data entries in cache sequentially accessed and a fourth data structure of data entries in cache non-sequentially accessed, wherein the third and fourth data structures have an MRU entry and an LRU entry;

determining whether the new data entry is sequentially accessed data;

indicating that the new data entry is at the MRU entry of the third data structure after determining that the new data entry is sequentially accessed; and

indicating that the new data entry is at the MRU entry of the fourth data structure after determining that the new data entry is non-sequentially accessed.

27. The article of manufacture of claim 21, further comprising a third and fourth data structures, wherein the first data structure indicates modified data entries written in cache and a storage unit as part of a sequential write operation, wherein the second data structure indicates modified data entries written in cache and the storage unit as part of a non-sequential write operation, wherein the third data structure indicates modified data entries written in cache and not the storage unit as part of a sequential write operation, and wherein the fourth data structure indicates modified data entries written in cache and not in the storage unit as part of a non-sequential write operation, wherein the step of processing the first and second data structures further comprises processing the third and fourth data structures to determine an LRU data entry from the first, second, third, and fourth data structures.

28. The article of manufacture of claim 27, further comprising the step of providing a modified data threshold indicating a maximum number of modified data entries to maintain in cache, wherein the step of processing the four data structures occurs after determining that the addition of modified data to the cache will cause the modified data threshold to be exceeded.

29. The article of manufacture of claim 28, further comprising a fifth data structure indicating the number of entries in the cache sequentially accessed and a sixth data structure indicating the number of entries in cache non-sequentially accessed, further comprising the steps of:

providing a sequentially accessed threshold indicating a maximum number of sequentially accessed data entries to maintain in cache;

determining whether the addition of a data entry to the cache will cause the number of data entries in cache to exceed the modified and sequentially accessed thresholds, wherein the step of processing the first, second, third, and fourth data structures occurs after determining that the addition of modified data to the cache will exceed the modified data threshold and not exceed the sequentially accessed threshold;

processing the fifth and sixth data structures to determine LRU data entries after determining that the modified

22

threshold is not exceeded, wherein the steps of selecting and demoting comprises selecting and demoting the LRU data entry that is the least recently used entry from the fifth and sixth data structures; and

processing the first and third data structures to determine LRU data entries after determining that the modified and sequentially accessed thresholds are both exceeded, wherein the steps of selecting and demoting comprises selecting and demoting the LRU data entry that is the least recently used entry from the first and third data structures.

30. The article of manufacture of claim 21, wherein the time value cannot exceed a maximum time value, wherein after reaching the maximum time value the time value resets to zero, further comprising the step of determining whether the time value for at least one of the LRU data entries was reset to zero, wherein the step of selecting the LRU data entry comprises selecting the LRU data entry that has a time value that was previously reset.

31. The article of manufacture of claim 21, wherein the time value comprises a sequence number that is incremented when data entries are indicated as added to the MRU entry of the data structures, wherein the maximum time value comprises a maximum sequence number, and wherein the LRU data entry in a data structure has the lowest sequence number in the data structure, wherein the step of determining whether the time value for an LRU data entry was reset to zero comprises the steps of:

(i) determining a difference between the sequence numbers of the LRU data entries in the first and second data structures; and

(ii) determining whether the difference between the sequence numbers is greater than half the maximum sequence number; and

wherein the step of selecting the LRU data entry from the data structures comprises the steps of:

(i) selecting the LRU data entry having a largest sequence number after determining that the difference between the LRU data entries is greater than half the maximum sequence number; and

(ii) selecting the LRU data entry having a lowest sequence number after determining that the difference between the LRU data entries is less than half the maximum sequence number.

32. The article of manufacture of claim 21, further comprising a third data structure indicating data entries in the first and second data structures, wherein each data entry in the third data structure has a modified time value indicating when the data entry was first modified in cache, further comprising the steps of:

executing a routine at predetermined intervals to destage data from cache to a storage unit;

providing a base time value indicating a previous time value;

determining, when executing the routine, data entries in the third data structure having a time value that is older than the base time value;

destaging, for each determined data entry in the third data structure, the copy of the modified data maintained in cache to the storage unit; and

indicating the destaged data entry as removed from the third data structure.

33. A memory area accessible to a processing unit, wherein the memory unit stores:

a plurality of data entries;

23

a first and second data structures indicating the data entries, wherein each data structure has a most recently used (MRU) entry, a least recently used (LRU) entry, and a time value associated with each data entry indicating a time the data entry was indicated as added to the MRU entry of the data structure; and

a new data entry added to the memory area, wherein the processing unit processes the first and second data structures to determine a LRU data entry in each data structure and selecting from the determined LRU data entries the LRU data entry that is the least recently used, wherein the processing unit demotes the selected LRU data entry from the cache and data structure including the selected data entry, and wherein the processing unit indicates that the new data entry as located at the MRU entry of one of the first and second data structures.

34. The memory area of claim 33, wherein the first and second data structures are comprised of linked lists in cache, wherein the MRU entry is at one end of the linked list and the LRU entry is at the other end of the linked list.

35. The memory area of 33, wherein the first data structure indicates data entries in cache sequentially accessed and the second data structure indicates data entries in cache non-sequentially accessed, wherein the processing unit determines whether adding the new data entry to the memory area would cause the number of sequentially accessed data entries to exceed a threshold, and wherein the processing unit selects the LRU data entry from the first data structure to demote after determining that adding the new data entry to cache would cause the sequentially accessed data entries to exceed the threshold.

36. The memory area of claim 35, wherein the processing unit indicates that a data entry to which a data access request was directed is located at the MRU entry of the first data structure after determining that the data access request is a sequential access, and wherein the processing unit indicates that the data entry to which the data access request was directed is located at the MRU entry of the second data structure after determining that the data access request is a non-sequential access.

37. The memory area of claim 33, wherein the first data structure indicates data entries written to the cache as part of a first type of write operation and the second data structure indicates data entries written to the cache as part of a second type of write operation, wherein the processing unit writes data written to the cache as part of the first type of write operation to a first storage unit, wherein the processing unit selects the LRU data entry from the first data structure to destage from the first storage unit to a second storage unit

24

after determining that the new data entry is part of the first write operation, and wherein the processing unit adds the new data entry to the first storage unit, wherein the step of indicating the new data entry as located at the MRU entry of one of the first and second data structures comprises indicating the new data entry as located at the MRU entry of the first data structure.

38. The memory area of claim 33, further comprising a third and fourth data structures, wherein the first data structure indicates modified data entries written in cache and a storage unit as part of a sequential write operation, wherein the second data structure indicates modified data entries written in cache and the storage unit as part of a non-sequential write operation, wherein the third data structure indicates modified data entries written in cache and not the storage unit as part of a sequential write operation, and wherein the fourth data structure indicates modified data entries written in cache and not in the storage unit as part of a non-sequential write operation, wherein the step of processing the first and second data structures further comprises processing the third and fourth data structures to determine an LRU data entry from the first, second, third, and fourth data structures.

39. The memory area method of claim 33, wherein the memory area further comprises a third data structure indicating data entries in cache sequentially accessed and a fourth data structure indicating data entries in cache non-sequentially accessed, wherein the third and fourth data structures have an MRU entry and an LRU entry, wherein the processing unit indicates that the new data entry is at the MRU entry of the third data structure after determining that the new data entry is sequentially accessed; and indicates that the new data entry is at the MRU entry of the fourth data structure after determining that the new data entry is non-sequentially accessed.

40. The memory area of claim 33, wherein the memory area further comprises:

a third data structure indicating data entries in the first and second data structures, wherein each data entry in the third list has a modified time value indicating when the data entry was first modified in cache; and

a base time value indicating a previous time value, wherein the processing determines data entries in the third data structure having a time value that is older than the base time value and, for each determined data entry in the third data structure, destages the copy of the modified data maintained in cache to the storage unit.

* * * * *



US006219751B1

(12) **United States Patent**
Hodges

(10) Patent No.: **US 6,219,751 B1**
(45) Date of Patent: **Apr. 17, 2001**

(54) **DEVICE LEVEL COORDINATION OF
ACCESS OPERATIONS AMONG MULTIPLE
RAID CONTROL UNITS**

(75) Inventor: **Paul Hodges, Los Gatos, CA (US)**

(73) Assignee: **International Business Machines
Corporation, Armonk, NY (US)**

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/069,070**

(22) Filed: **Apr. 28, 1998**

(51) Int. Cl.⁷ **G06F 12/14; G06F 13/14**

(52) U.S. Cl. **711/114; 711/154**

(58) Field of Search **711/111, 112, 114,
711/154, 155**

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,207,609 6/1980 Luiz et al. 364/200

4,761,785 8/1988 Clark et al. 371/51
4,914,656 4/1990 Dunphy, Jr. et al. 371/10.2
5,265,098 11/1993 Mattson et al. 371/11.1
5,278,838 1/1994 Ng et al. 371/10.1
5,530,948 * 6/1996 Islam 395/182.04

* cited by examiner

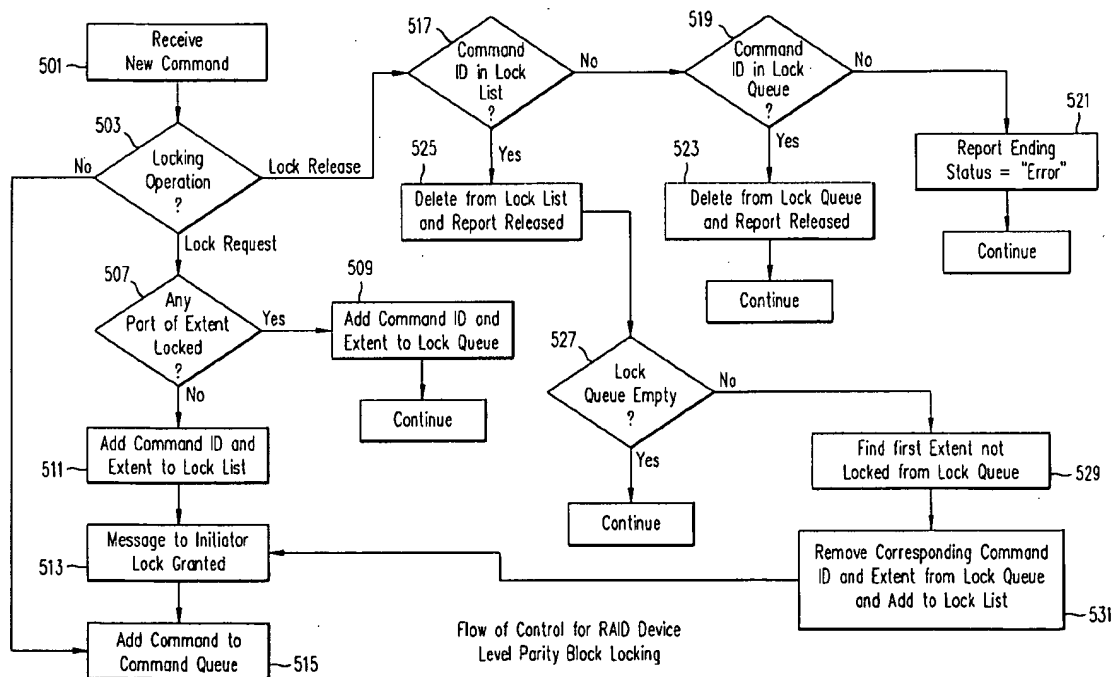
Primary Examiner—Kevin L. Ellis

(74) Attorney, Agent, or Firm—R. Bruce Brodie; Randall J.
Bluestone

(57) **ABSTRACT**

A method and apparatus for serializing access to disk arrays shareable among a plurality of RAID control units at a substantial reduction in intercontrol unit communication by (a) defining a lock function over the parity image blocks at each of the disk drives of a shared disk array; and (b) executing a path expression at each accessing control unit, the path expression includes requesting a lock from the drive on the parity image and enforcing a busy-wait until a lock is granted, executing the RAID function, and then releasing the lock.

19 Claims, 5 Drawing Sheets



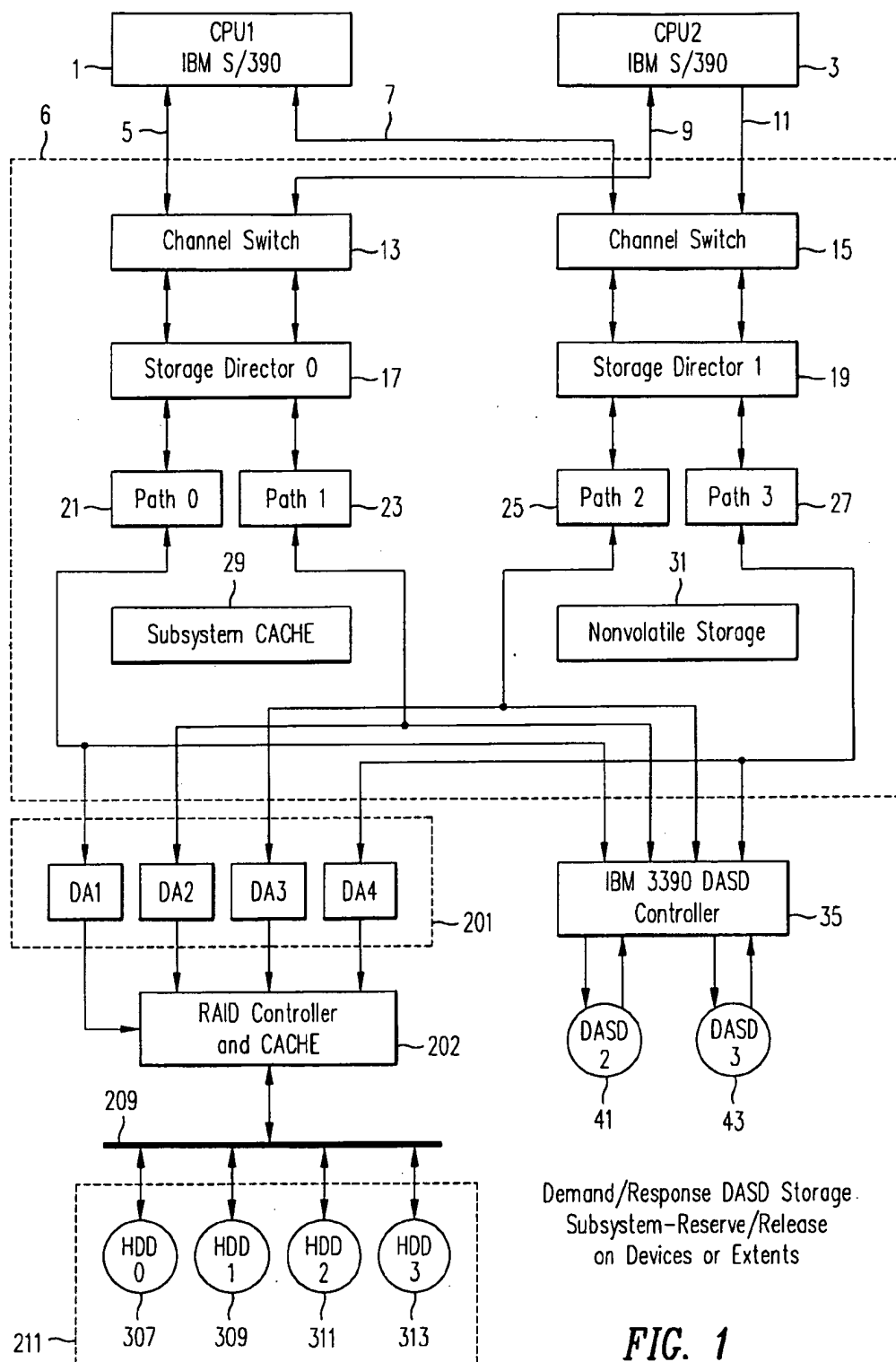
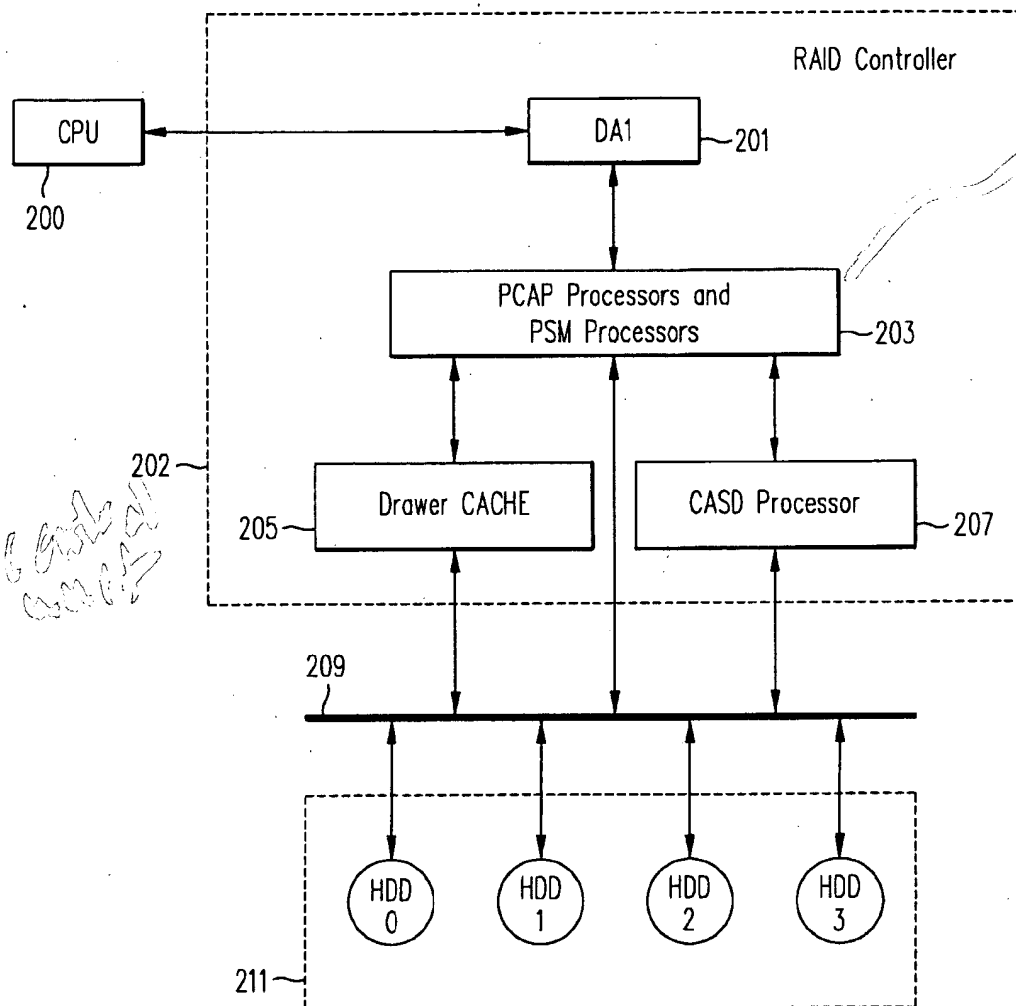
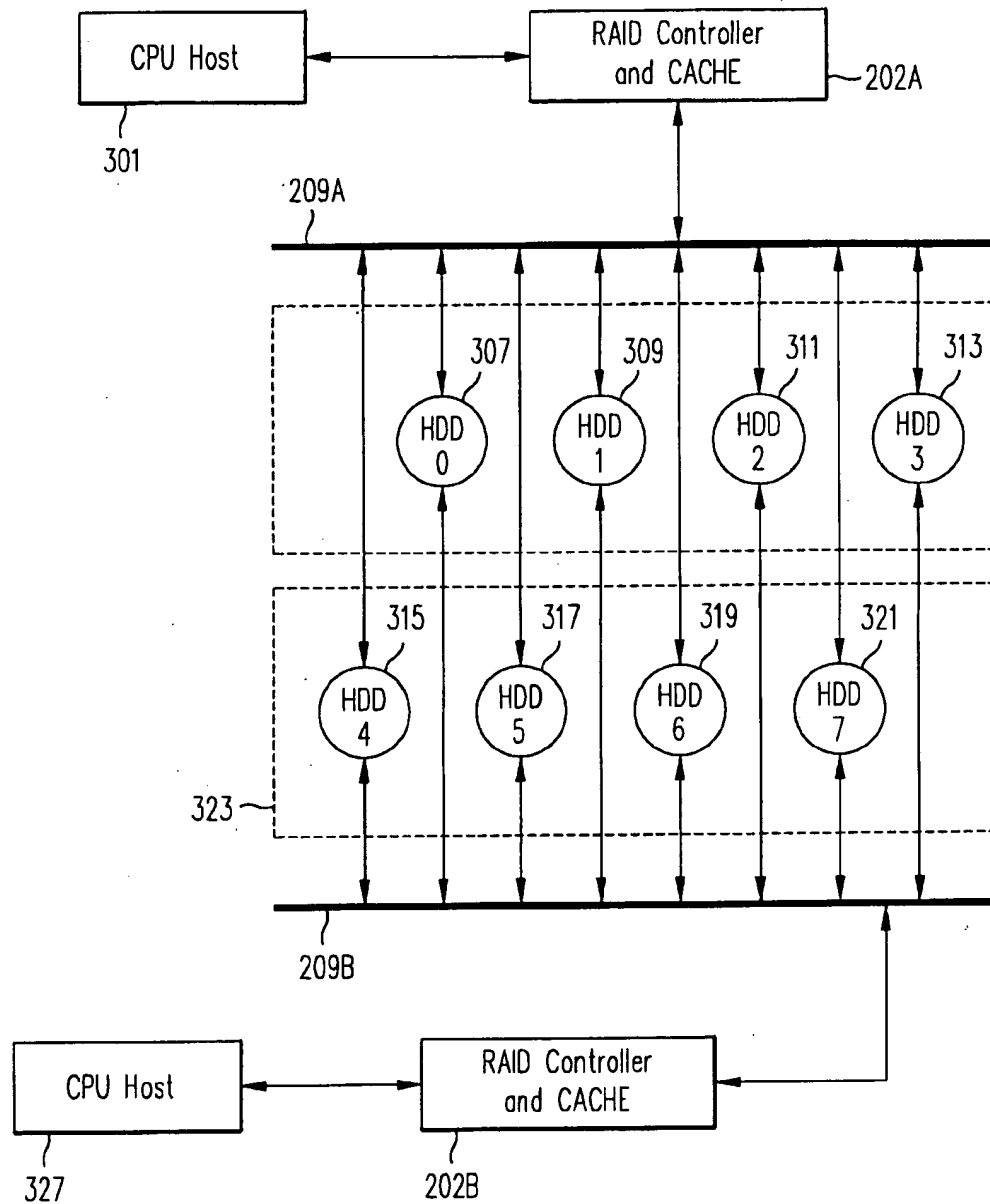


FIG. 1
(Prior Art)



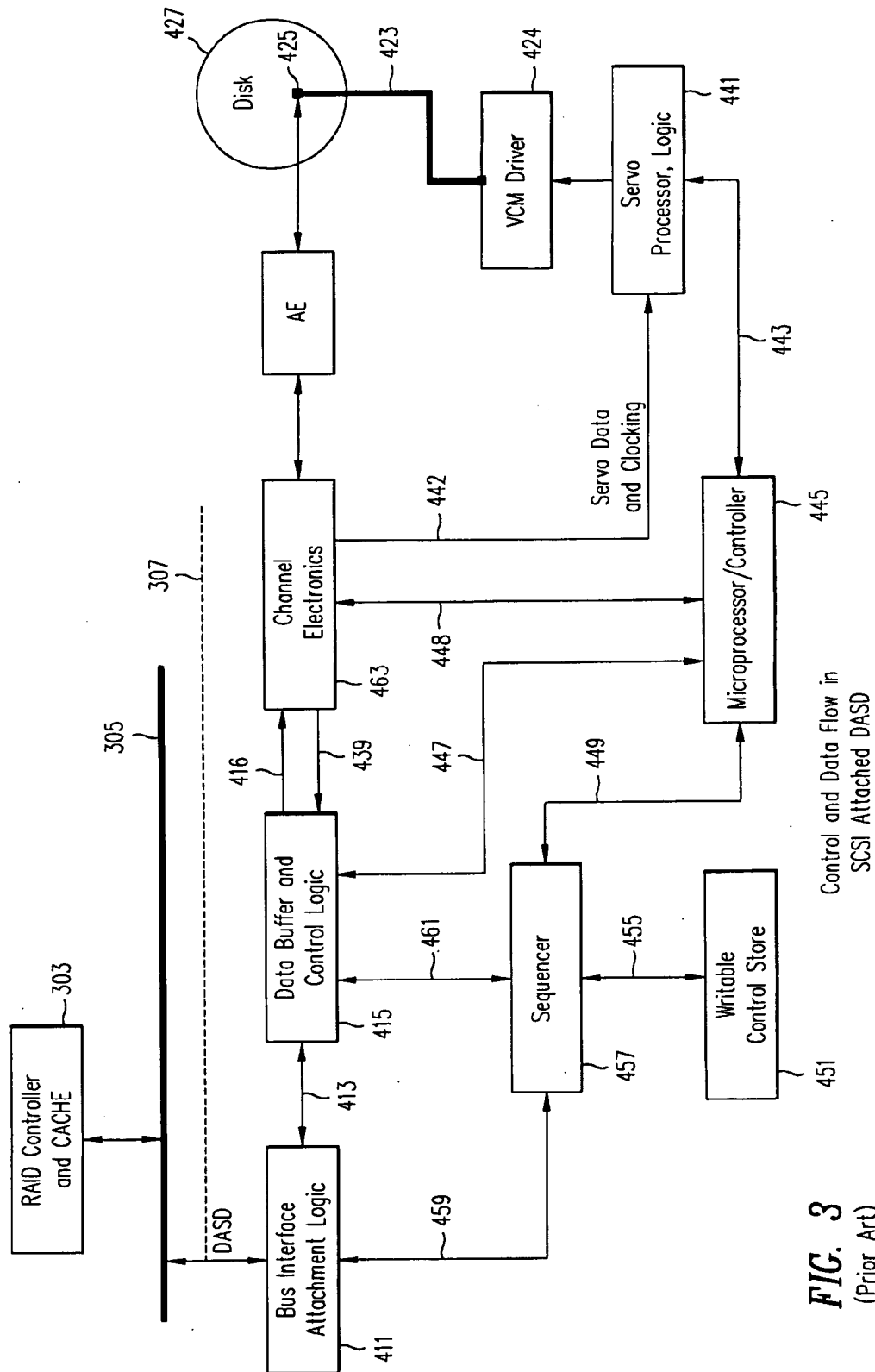
RAID Array with a Single Controller

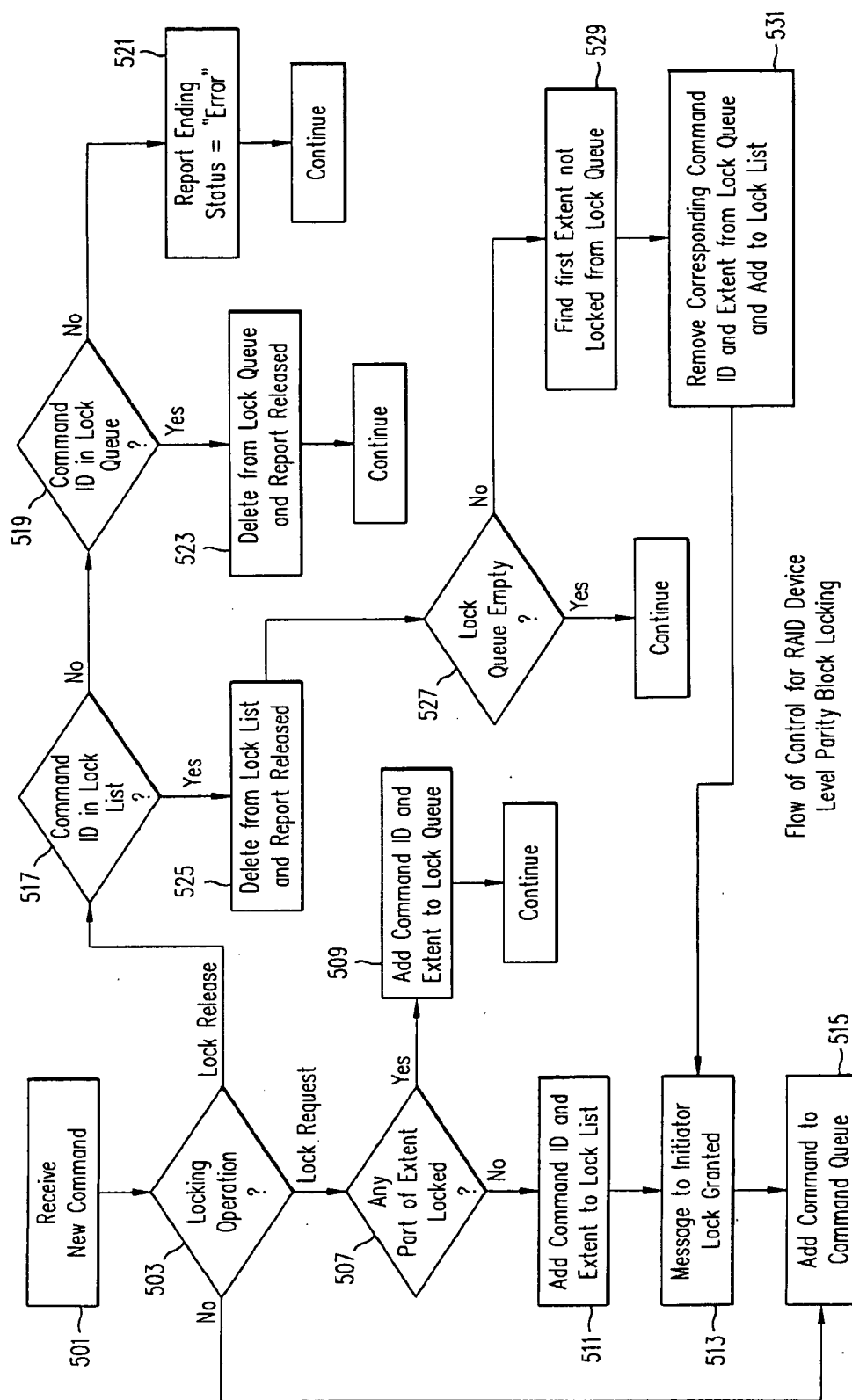
FIG. 2A



Multiple Arrays with Dual RAID Controllers

FIG. 2B





Flow of Control for RAID Device
Level Parity Block Locking

FIG. 4

1

DEVICE LEVEL COORDINATION OF ACCESS OPERATIONS AMONG MULTIPLE RAID CONTROL UNITS

FIELD OF THE INVENTION

This invention relates to storage subsystems formed from cyclic, multitracked devices, and more particularly to storage subsystems of the RAID 3 or RAID 5 type in which multiple RAID control units share access to arrays of such cyclic devices.

DESCRIPTION OF RELATED ART

The succeeding paragraphs briefly describe the data availability, storage capacity, and data rate tradeoffs among several RAID disk array configurations. This is followed by a discussion of aspects of the prior art management of accessing disk drives and arrays shared among two or more control units.

RAID Arrays Fault Tolerance, and Recovery

In the prior art, RAID arrays of cyclic, tracked storage devices have found use in increasing the availability and reliability of stored data. The increased availability and reliability is achieved by dedicating a portion of the capacity to redundant information. In the presence of corrupted data or unavailable disk drives, the RAID control unit uses the redundancy to either reconstruct data on the fly, rebuild data on spare disks, or both. Thus, in a RAID 1 configuration, each update to data is written out to two disks. If any single disk fails, then the duplicate disk is electronically switched into the access path as its replacement. Although remarkably fault tolerant, an N disk RAID 1 array has a storage capacity limited to N/2 of its drives.

A RAID 3 configuration is data rate intensive and sustains a data rate N times the rate of a single disk. Also, it creates a logical track N times the size of a physical track. In RAID 3, N data blocks at a time are written or read across N counterpart synchronized disks and a parity image on an N+1st drive. Unfortunately, the high data rate also means that the concurrency rate is low. That is, only one application can access the drives.

In contrast, a RAID 5 configuration is transaction or concurrency intensive. As illustrated in Clark et al., U.S. Pat. No. 4,761,785, "Parity Spreading to Enhance Storage Access", issued Aug. 2, 1988, N-1 data blocks and an associated parity image are written across N asynchronous disk drives in the same physical address range such that no single drive stores two or more blocks from the same parity group, and such that no single drive stores all of the parity blocks.

It should be recalled that the RAID 3 configuration is affected by adverse loading. Each read and write requires that all drives be accessed, including the parity drive. However, in the RAID 5 context, adverse loading can be minimized in several ways. First, since parity is spread out among the disks, no single disk bears all the parity loading. Indeed, Mattson et al., U.S. Pat. No. 5,265,098, "Method and Means for Managing DASD Array Accesses When Operating in Degraded Mode", issued Nov. 23, 1993, proposed spreading data and parity blocks out among the disks in a pattern forming a balanced incomplete block design such that adverse loading would be minimized, even where a disk failed and the array was operating in a fault-tolerant mode.

There have been many proposals both for operating RAID 5 arrays and the like in degraded mode and returning the information state of any given array back to a fault-tolerant

2

mode. In this regard, Dunphy, Jr. et al., U.S. Pat. No. 4,914,656, "Disk Drive Memory", issued Apr. 3, 1990, describe the use of a pool of hot spare disks available for rewriting in the event of single disk failure and in the presence of single image parity groups.

Similarly, Ng et al., U.S. Pat. No. 5,278,838, "Recovery From Errors in a Redundant Array of Disk Drives", issued Jan. 11, 1994, disclose the online scheduling and rebuilding of data on a spare DASD or the like, the data having been stored on unavailable disk drives in a type RAID 1, 4, or 5 array. The Ng invention relies upon coded error detection and assumes that failures would occur as random independent events.

Management of Accessing Among RAID Control Units and Shared Disk Arrays

In a RAID 5 disk array, updating one or more of the N blocks in a stripe stored on N disks requires four or more access operations and the recalculation of the parity block image. The data and parity blocks move between a control unit resident cache or buffer and two or more of the disk drives. The operations consist of (1) reading the old data block from disk; (2) reading the old parity block from disk; (3) writing the new data block to disk, recalculating new parity as the XOR of the old parity, old data, and the new data; and (4) writing the new parity block to disk.

When two or more RAID control units seek access to data on drives in a shared array, there are myriad opportunities to corrupt data, such as where RAID read and write operations occur concurrently and where they involve more than one drive. Avoidance requires that RAID operations be coordinated in order to preserve data integrity. In the prior art, the RAID control units have been required to communicate and negotiate a lock-like state serializing their access to the disk data. The constructs for serializing access between the RAID control units would frequently be some form of shared variable or message passing. Shared variable synchronization includes test and set, semaphores, conditional critical regions, monitors, or path expressions.

While any one of these constructs, when utilized by both control units, serialize access to the same resource and preserve data integrity, they require that each RAID control unit devote considerable bandwidth to originating and sending messages as well as receiving and interpreting messages from the other control unit. Given that RAID control units may be configured in network relationships, the message traffic required to synchronize access to shared disk arrays becomes nonlinear. Cumulatively, the network bandwidth dedicated to synchronizing communications reduces the bandwidth available to either data rate or concurrency, or other storage and data management tasks.

SUMMARY OF THE INVENTION

It is accordingly an object of this invention to devise a method and apparatus for serializing access to disk arrays shareable among a plurality of RAID control units at a substantial reduction in intercontrol unit communication.

It is yet another object that such method and means effectuate serialized access to data on said devices independent of any consistent state as among concurrently accessing RAID control units.

It is a related object that such method and apparatus minimize reduction in either data rate or concurrency as a function of differences among random or sequential access patterns.

The above objects are believed satisfied by a method and apparatus for selectably locking a transfer path between one

of several RAID control units to a shared array of cyclic, multitracked storage devices (disk drives). The control units perform RAID functions on at least one data block and a parity image of a parity set defined over the disk drives. The blocks are distributed such that no single disk drive in the array stores more than one block from an associated data set.

The method steps include defining a lock function on each disk drive. This enables the disk drive to enqueue lock requests from accessing control units, grant lock requests to a requesting drive in enqueued order to an available parity image stored on the device, and to release the lock responsive to a control unit signal. The next step involves executing at the control unit a RAID function embedded within a path expression, a path expression being a construct for synchronizing and ordering the activity relationship between the control unit and one or more array drives. In this regard, the RAID function requests an explicit lock on at least one predetermined parity image from a counterpart device. Significantly, the execution of any RAID function is inhibited until grant of the lock by the counterpart disk drive as a form of busy-wait. When a lock is granted by the disk drive, the control unit then proceeds to execute the RAID function on at least one data block and parity image of an associated parity set from the counterpart devices. The control unit signals the drive upon termination of the RAID function, thereby causing the device to relinquish the lock.

In this invention, the RAID control units are preferably of a RAID 3 type and RAID 5 type or combination thereof. Both types satisfy the constraint that no single drive stores more than one block from the same parity set. Additionally, a RAID 5 array is further limited in that no single drive stores all the parity images for the parity sets defined onto the array.

A path expression comprises a composite function executed at a control unit specifying an ordering of uninterruptible procedures interpreted by at least one of the devices. It forms a corouting relationship with at least one of the storage devices. The ordering includes effectuating lock access to the parity image on the counterpart device, executing a RAID function over at least one data block and the parity image, and relinquishing the lock access. Advantageously, each path expression executed at a first control unit is independent of the consistency state of every other RAID control unit.

In this invention, a RAID function is selected from a set consisting of reading, writing, or write modification of data blocks and parity image of a parity set, data block or parity image regeneration, and parity set rebuilding. Typically, such RAID functions include multiple read and write commands of which any lock request would be embedded in a first read or write command in the RAID function.

BRIEF DESCRIPTION OF THE DRAWING

FIG. 1 shows a logical block diagram of a RAID 5 disk array in a hierarchical storage subsystem according to the prior art.

FIGS. 2A-2B respectively depict a RAID-organized disk storage subsystem directly attached to a single host and a configuration where at least two disk arrays are attached between two or more RAID control units.

FIG. 3 sets out the control and data flow of the command and data transfer paths of a typical SCSI disk drive array attached as in FIGS. 2A and 2B.

FIG. 4 illustrates the flow of control for RAID device level parity block locking according to the invention.

DESCRIPTION OF THE PREFERRED EMBODIMENT

Preliminary Comments and Clark as a RAID Array Paradigm

Before describing the preferred embodiment, several preliminary matters should be considered. First, in this preferred embodiment, the RAID arrays of choice are of the RAID 3 and RAID 5 types. Second, the general attributes of RAID device types 1-5 are fully described in the Clark et al. patent and the references cited therein. Third, the term "parity set" is any logical association of N data blocks and a parity image taken thereover as the $(N+1)^{th}$ block. Fourth, the terms "logical block" and "block" are used synonymously and refer to a fixed length of addressable storage extent used on a disk drive. Fifth, the term "enqueue" connotes the operation of placing a resource request or command in a queue or waiting list. Classically, enqueueing connotes a pair of synchronization primitives. The first primitive "enqueue" is the operation of placing a resource request in a queue under some discipline, such as FIFO. The request waits until the resource becomes available. At this point, the second primitive "dequeue" is invoked as the operation of placing the resource under the control of the requesting source and removing it from the queue or list.

In both RAID 3 and RAID 5 arrays, there are several ways of creating or defining parity sets over the disks. One convenient method is to define them by their storage proximity of blocks on disk drives as in the aforementioned Clark '785 patent or in the Mattson '098 patent.

In Clark, a parity image is formed from the data that is stored in the same range R_1 of contiguous physical addresses in each of N disks with the parity image also being stored in the range R_1 on the $N+1^{st}$ disk. The $N+1$ blocks collectively are termed a "stripe" or a parity set. In Clark, the location of the parity image is rotated or spread in a round-robin manner from stripe to stripe. From Clark, two logical relations can be discerned. First, the data blocks in the same parity set or stripe are resident in the same address range R_1 on different disks. Second, the data blocks are covered by the same parity image. A similar convention is expressed in Mattson.

Serialization of Access to a Shared Resource

In the past, concurrent access to shared disk data was managed either by high-level CPU lock-oriented serialization or significant intercontrol unit communications supporting a busy-wait condition. The embodiment in FIG. 1 illustrates a typical RAID 5 array as a substitute large logical disk positioned in a hierarchical storage subsystem with no intercontrol unit communications.

Referring now to FIG. 1, there is shown a logical block diagram of a RAID 5 disk array in a hierarchical storage subsystem according to the prior art. The RAID array 202, 211 is accessed by applications executing either on multi-tasking hosts CPU 1 or CPU 3, such as an IBM System/390 running under the IBM MVS operating system. The access is imposed over a path including the storage control unit 6 exemplified by an IBM 3990 SCU Mod 6. In this configuration, the RAID array 202, 211 is externally operated as a storage endpoint in much the same manner as conventional disk drives 41 and 43 (such as the IBM 3390) under control unit 35.

The subsystem depicted in FIG. 1 is designed such that data stored on any of the disk storage devices 211, 41, and 43 can be accessed over any one of at least two failure-independent paths from either one of the CPUs 1 or 3, although the system as shown provides four failure-independent paths. Illustratively, data on devices 211 can be reached over any one of paths 21, 23, 25, or 27. The same holds for data stored on devices 41 or 43 via control unit 35. A full description of this principle is to be found in Luiz et al., U.S. Pat. No. 4,207,609, "Method and Means for Path Independent Device Reservation and Reconnection in a Multi-CPU and Shared Device Access System", issued Jun. 10, 1980.

5

In FIG. 1, when two applications on the same or different CPUs seek concurrent access to the same data sets within the same data volume, the condition in MVS is resolved through the use of RESERVE/RELEASE or similar commands. Under the MVS operating system, RESERVE is a command that turns a hardware lock against an entire disk volume so that no other CPU may access it. For reasons previously mentioned, such high-level inspired serialization involves significant CPU processing overhead, denies path access to significant amounts of disk-stored data, and increases susceptibility to deadlock. The latter occurs when applications on CPU 1 and CPU 3 each reserve two separate disk volumes, and then each requests the volume that the other has reserved.

Single and Shared RAID Disk Arrays

Referring now to FIG. 2A, there is shown a RAID 5 disk array organized storage subsystem directly attached to a single host CPU 200. In this configuration, a RAID 5 subsystem includes a control unit 202 and four attached disk drives 211, otherwise denominated as in FIG. 1 as disks 307, 309, 311, and 313 such as may be found in an IBM RAMAC Array DASD attaching one or more Enterprise System (S/390) ECKD channels through an IBM 3990 Mod 3 or 6 storage control unit. The RAMAC array disk storage subsystem comprises a rack with a capacity between 2 to 16 drawers. Each drawer includes four disk drives HDD0-HDD3 (211) and a control unit 202. The RAID control unit or control unit 202 includes cooling fans, control processor 207, ancillary processors 203, and a nonvolatile drawer cache 205.

Functionally, a device attachment unit 201 provides electrical and signal coupling between the CPU 200 and one or more RAID 5 drawers. As tracks are staged and destaged through this interface, they are converted from variable-length CKD format to fixed-block length FBA format by the ancillary processors 203. In this regard, drawer cache 205 is the primary assembly and disassembly point for the blocking and reblocking of data, the computation of a parity block, and the reconstruction of blocks from an unavailable array disk drive. A typical configuration would consist of several drawers. An additional drawer (not shown) would include four disk drives operable as "hot spares". This is an alternative to siting a "hot spare" within each of the operational drawers.

In this embodiment, the four disk drives 307-313 are used for storing parity groups. If a dynamic (hot) sparing feature is used, then the spare must be defined or configured a priori in the spare drawer. Space among the four operational array devices is distributed such that there exists three DASD's worth of data space and one DASD's worth of parity space. It should be pointed out that the disk drives 211, the cache 205, and the processors 203 and 207 communicate over a SCSI-managed bus 209. Thus, the accessing and movement of data across the bus between the disk drives 211 and the cache 205 is closer to an asynchronous message-type interface. A typical layout of CKD tracks and parity images of groups of CKD tracks over the disk drives follows the pattern described in the description of the prior art with reference to the Clark '785 patent.

Referring now to FIG. 2B, there is shown a configuration where at least two disk arrays 211 and 213 are attached between two or more RAID control units 202 A and 202 B via SCSI buses 209A and 209B. Alternatively, two or more disk drives could be multidropped between the pair of buses. For purposes of this discussion, the arrays are logically organized such that array 211 comprises disks 307-313 and array 213 comprises disks 315-321. As pointed out in the

6

discussion of the embodiment of FIG. 1, each disk drive is at least dual ported. Thus, either RAID control unit may access every one of the drives in either disk array.

The problem confronted in this invention is where CPU 301 and CPU 327 concurrently request access to drives within the same parity set or stripe. Indeed, the shared access conflict is expected to increase over time as client/server and network models proliferate.

Disk Drive Execution Environment

The solution broadly requires (a) defining a lock function over the parity image blocks at each of the disk drives of a shared disk array; and (b) executing a path expression at each accessing control unit, the path expression includes requesting a lock from the drive on the parity image and enforcing a busy-wait until a lock is granted, executing a RAID function, and releasing the lock. This requires that each drive have sufficient processing and local memory capability. In this regard, reference should be made to FIG. 3.

Referring now to FIG. 3, there is shown the control and data flow of the command and data transfer paths of a typical SCSI disk drive, such as would be used in arrays 211 or 213. The drive is organized around two processing paths, namely a command processing path and data transfer path. The command processing path includes a bus interface 411, a sequencer 457, a microprocessor 445, and a servo processor 441. The data transfer path includes the bus interface 411, a data buffer 415, channel electronics 463, a read/write head 425, and a cyclic, multitracked disk 427.

In FIG. 3, data is streamed out to or derived from addressed tracks on the magnetic or optical disk 427 over the data path, while storage (read/write) and access (seek/set sector) commands are processed by a command path also within the disk drive 307. Commands and data from the host 1 are passed through the interface 411. As suggested, the commands are interpreted and processed over the path including a sequencer 457, the microprocessor control unit 445, servo processor logic 441, and the physical accessing mechanism 423-425 to the cyclic, tracked disk 427. In contrast, data is passed to or from tracks on the disk 427 via the interface 411, a data buffer 415, channel electronics 463, a read/write head 425 adjacent to the recorded data on the track, and amplifier electronics 422.

The above-mentioned lock facility can be expressed at the disk drive 307 as a series of functions written into control store 451. Appropriate lock constructs, such as a lock table or list-lock queue and command queue, are maintained at the microprocessor 445. The thesis of this invention is that the addition of appropriate locking functions and queues of lock requests in the disks drives can be used to coordinate the functions of accessing RAID control units to both maintain data integrity and eliminate intercontrol unit communication. More particularly, drive-level locking combined with executing access operations as path expressions (composite functions) will avoid conflicting operations between the control units.

Lock Commands

Implementation of any lock facility requires defining four additional device commands, namely, Read/Lock, Write/Lock, Lock, and Release/Lock.

The Read/Lock command is issued by the control unit to a disk drive in an array storing a parity image addressed by the command. The Read/Lock command also recites a range of logical block addresses (LBAs) which prospectively are involved in update, rebuild, or regeneration operations. A command tag serves as an identifier for later release of the lock.

The Read/Lock command can be processed in one of two ways. A first approach is for a control unit to send only the lock request and delay sending the read command until after receipt of the lock grant by the control unit from the disk drive. A second approach is for the control unit to send both parts of the command to the disk drive. If any part of the LBA range is already locked when the disk drive receives the Read/Lock command, the lock request as represented by this command will be enqueued at the disk drive until any previous lock on this LBA range is released. As soon as the lock becomes available, the drive can execute the read command. The first approach places the onus on the control unit, while the second passes the same to the disk drive.

The Write/Lock command is issued by the control unit to a disk drive in an array with a range of LBAs for a full parity set or stripe write. The lock request and write delay is processed by the control unit and the disk drive in the same manner as is used in processing the Read/Lock command. However, after the lock has been granted, the command is executed as a normal write operation with respect to designated data blocks and the parity image as reflected in the LBA address range designation. As mentioned above, the use of a single command allows earlier queuing of the write operation at the disk drive.

The Lock command is issued by the control unit to a disk drive in an array storing a target parity image with a range of LBA addresses for either a partial stripe Write or a full or partial stripe Read. The lock portion is implemented in the same manner as that of the Read/Lock command. In this command set, there is no implied read or write operation. Any read or write command to the parity drive must be issued or given effect only after the lock has been granted.

The Release/Lock command is issued with an appropriate identifier (initiator and tag) when the sequence for which the lock was issued is complete. This also pertains if the sequence must be canceled. In this regard, the Release/Lock command operates to cancel the corresponding lock. The command is rejected if there is no lock in effect or queued with that identifier.

Illustrative Path Expressions with Included RAID Functions

As previously mentioned, a path expression is a synchronization construct to secure an interference-free interaction between a control unit and a drive storing a target parity image. Other terms, such as composite function or composite operation, may be used synonymously.

Three illustrative expressions set out below are Update Data, Rebuild Data, and Regenerate Data. The expressions are set out in a pseudo-code-like format and are executable by any of the control units such as 202A or 202B in FIG. 2B with respect to a disk drive 307 in array 211.

I—Update Data

(a) Issue a Read/Lock command with a predetermined tag and LBA range to the parity disk drive.

(b) When an indication has been received from the disk drive that the lock has been granted, issue the Read command (this constitutes reading of old data for Update Write purposes).

(c) When the old data block has been read, issue a Write command to the disk drive storing the old data block to update the data in place.

(d) When the old parity has been received by the control unit from execution of the Read/Lock command at the parity disk drive, generate a new parity and issue a Write command to the parity drive to update in place the parity image block.

(e) When the control unit receives indication that both Write commands have been executed, issue a Release/Lock command to the parity drive.

II—Rebuild Data

(a) Issue a Read/Lock with a tag and LBA range to the parity drive.

(b) When an indication has been received from the disk drive that the lock has been granted, issue a Read command to each of the data drives in the target parity set.

(c) When indications have been received that all of the Read commands have been executed, issue a Write command to the drive being rebuilt.

(d) When indication has been received that the Write command has been completed, issue a Release/Lock command to the parity drive.

III—Regenerate Data

(a) Issue a Read/Lock with a tag and LBA range to the parity drive.

(b) When an indication has been received from the disk drive that the lock has been granted, issue a Read command to each of the data drives in the target parity set.

(c) When indications have been received that all of the Read commands have been completed, issue a Release/Lock command to the parity drive.

(d) Transfer regenerated data to the host CPU independent of the disk drive operation.

Lock Constructs and Processing at the Disk Drives

It was pointed out that contemporary disk drives, such as depicted in FIG. 3, include significant local operation scheduling and processing capacity. For this reason, each disk drive will include several lock management constructs sited in its local processor memory or equivalent. These constructs include a Command Queue, a Lock List, and a Lock Queue.

A Command Queue is an ordered list of executable commands received by a disk drive and not yet executed. Executable commands are those commands that inherently either do not require locks or that have locks in effect. Commands in the Command Queue may be reordered by the disk drive to optimize performance.

A Lock List is a list of granted locks currently in effect. A lock list associates a command identification with a corresponding storage address extent. Significantly, address extents must not overlap.

A Lock Queue is an ordered list of lock requests or pending locks. Each lock request or lock queue entry associates a command identification with a corresponding extent of disk drive addresses. When a preexisting lock is released, the lock discipline requires that the list be searched for the first command for which a new lock can be granted.

Referring now to FIG. 4, there is shown the flow of control for RAID device level parity block locking according to the invention. Each new command is received by a disk drive in step 501 and assessed in step 503 as to whether it is a lock release, a lock request, or none of the above. If it is a lock release, control is transferred to step 517. If it is a lock request, control transfers to step 507. If none of the above, then the process jumps to step 515 where the command is added to the command queue.

If the command is a lock request, step 507 determines whether any part of the LBA address range as recited in the new command is already under lock. If the address range is currently subject to another lock, then the new command identification (ID) and the address extent are added to the lock queue in step 509. On the other hand, if the extent is not under lock, the new command ID and extent are added to the lock list in step 511. At some point time subsequent, the disk drive grants the lock in step 513 and adds the command ID and extent to the command queue.

If the command as tested in step 503 is a lock release, then it is examined in step 517 as to whether the command ID and

extent are already in the lock list. The presence of the command ID in the lock list will cause it to be deleted and the lock released in step 525. The lock queue will be tested for empty in step 527. If it is not empty, the lock queue is searched in step 529 and the first extent and associated command ID that are not locked are removed from the lock queue and added to the lock list in step 531. This further results in a lock being granted thereon in step 513 and the command added to the command queue in step 515.

For completeness, it should be said that if a lock release is not in the lock list per step 517 and is not in the lock queue per step 519, then an error is reported in step 521. However, if it is in the lock queue per step 519, then it is deleted from the queue per step 523.

Example of the Lock Processing

Suppose a series of commands have been received by disk drive 307 from a RAID controller in a predetermined order. Each command is associated with a command ID and an extent or range of addresses over which the command will operate. The command order is stipulated as follows:

TABLE 1

Command ID	LBA Extent	Lock Status of LBA Extent
A1	1000-1500	Yes-effected
B1	5000-6000	Not required
A2	3000-3100	Yes-effected
A3	1300-1600	Yes-queued until release by A1
A4	6500-7000	Not required
A5	3000-3500	Yes-queued until release by A2 and B2
B2	3200-4000	Yes-effected
B3	2900-3100	Yes-queued until release by A2

The processor within the effected disk drive will form the lock constructs of a Command Queue, a Lock Queue, and a Lock List based on this example and the above definitions.

The Command Queue would be:

TABLE 2

Command ID	LBA Extent
A1	1000-1500
B1	5000-6000
A2	3000-3100
A4	6500-7000
B2	3100-4000

The Lock Queue would be:

TABLE 3

Command ID	LBA Extent
A3	1300-1600
A5	3000-3500
B3	1900-3100

The Lock List would be:

TABLE 4

Command ID	LBA Extent
A1	1000-1500
A2	3000-3100
B2	3200-4000

Referring again to FIG. 4 together with Tables 1-4, it should be apparent that in processing the first entry in Table 1, say command A1, it would be identified as a lock request

in step 503. Tracing it further, it would also be apparent that no part of its LBA extent was under lock per step 507. Consequently, command A1 would be added to the lock list in step 511 (Table 4) and the lock granted in step 513. Lastly, it would be added to the command queue (Table 2) in step 515.

In contrast, command A3, while constituting a lock request in step 503, does have a portion of its LBA extent subject to a lock under A1 in step 507. Accordingly, it is entered into the lock queue (Table 3) in step 509.

Commands B1 and A4 do not require locks and are moved directly onto the command queue (Table 2) per steps 503 and 515.

The processing of a lock release is not treated directly in the above example. However, command A5 will remain on the lock queue (Table 3) until locks associated with commands A2 and B2 are released. The lock release of A2 and B2 respectively by a control unit would be processed in a traverse including steps 503, 517, 525, 527 and 529 since the lock queue (Table 3) would not be empty.

While the invention has been described with respect to an illustrative embodiment thereof, it will be understood that various changes may be made in the method and means herein described without departing from the scope and teaching of the invention. Accordingly, the described embodiment is to be considered merely exemplary and the invention is not to be limited except as specified in the following claims.

What is claimed is:

1. A method for serializing access to individual storage devices in an array of storage devices, said array having parity groups of data blocks and an associated parity image block written across counterpart storage devices such that no drive stores more than one block of the same group, said array being addressable by two or more control units, comprising the steps of:

(a) defining a lock function over the parity image blocks at storage devices of the storage device array; and

(b) executing a path expression at an accessing control unit, the path expression includes requesting a lock from the storage device storing the parity image of the parity group addressed by the control unit, and enforcing a busy-wait until a lock is granted by the storage device, executing a RAID function, and then releasing the lock.

2. The method according to claim 1, wherein the RAID function is one selected from a set consisting of modifying at least one data block and the parity image from at least one parity set on at least one device, and reconstructing at least one given data block or given parity image from remaining data blocks or parity image of any given parity set in the event of the unavailability of the given data block or parity image due to noise, corruption, or device failure and either staging it to a requesting control unit, writing it to at least one spare device or reserved area on the plurality of devices, or both.

3. The method according to claim 1, wherein the path expression comprises a composite function specifying an ordering of uninterruptible procedures interpreted by at least one of the devices, the ordering includes effectuating lock access to the parity image on the counterpart device, executing a RAID function over at least one data block and the parity image, and relinquishing the lock access.

4. The method according to claim 1, wherein each path expression being executed at a first control unit is independent of a consistency state of a second control unit.

5. The method according to claim 1, wherein the step of requesting a lock from the device includes the step of

11

generating a lock-oriented command selected from a command set consisting of Read/Lock, Write/Lock, Lock, and Release/Lock, each lock-oriented command including a tag operable as an identifier for subsequent release of any granted lock.

6. The method according to claim 1, wherein requesting a lock from the storage device further comprises requesting a lock of a parity group corresponding to a range of Logical Block Addresses (LBAs) requested by the control unit, and wherein enforcing a busy-wait further comprises enforcing a busy-wait until a lock of the parity group corresponding to the range of LBAs requested by the control unit is granted.

7. The method according to claim 1, wherein each device has defined thereon one data block from a first parity set and a parity image of a second parity set, and wherein the method further comprises the steps at the device of enqueueing any lock request embedded in a first read or write command embedded in a RAID function, said device being responsive to at least one of a sequence of commands from a control unit executing a path expression during the pendency by the device of any current lock.

8. The method according to claim 7, wherein the RAID control units are of a type selected from a set consisting of type RAID 3 and type RAID 5 control units, and further wherein the RAID function is selected from a set consisting of reading, writing, or write modification of data blocks and parity image of a parity set, data block or parity image regeneration, and parity set rebuilding.

9. The method according to claim 7, wherein the step of executing a path expression includes the step of forming a corouting relationship with at least one of the storage devices.

10. The method according to claim 7, further comprising terminating the RAID function which includes signaling the device upon completion of all read and write commands within the RAID function.

11. The improvement according to claim 10, wherein said first circuits include circuits for ascertaining whether a parity set covered by a lock request is concurrently subject to a lock in whole or in part and if subject to a lock said lock request is enqueued, and if not subject to a lock said lock request is granted and any command associated with said request in the path expression is placed on a command queue for execution by the device.

12. A method for establishing a locked path between one of a plurality of RAID control units and at least one of a plurality of cyclic, multitracked storage devices, said control units concurrently accessing selected blocks of data from parity imaged data sets defined over the devices, the blocks being distributed such that no single device stores more than one block from an associated parity imaged data set, comprising the steps at each of the RAID control units of:

(a) defining a facility at each of the devices for enqueueing lock requests from said control units and for granting, maintaining, and releasing locks on the parity image resident on the device, said locks being granted in enqueued lock request order; and

(b) executing a path expression at respective control units inclusive of a RAID function including:

(1) requesting a lock from the device on the parity image of an associated set, at least one of whose blocks is addressed in a counterpart RAID function; and
(2) inhibiting execution of any RAID function with respect to any parity image or data blocks addressed by said RAID function until a lock is granted by the device storing the parity image to the counterpart RAID control unit;

12

(3) executing the RAID function with reference to the parity image and data blocks addressed by the request responsive to the grant of the lock; and
(4) terminating the RAID function and causing release of said lock by the granting device.

13. A method for path locking one of a plurality of RAID control units to at least one of a plurality of shareable cyclic, multitracked storage devices, the control units performing RAID functions on at least one data block and a parity image of a parity set defined over the devices, the blocks being distributed such that no single device stores more than one block from an associated data set, comprising the steps of:

(a) defining a lock function on each device and causing said device to enqueue lock requests from accessing control units, grant lock requests to a requesting device in enqueued order to an available parity image stored on the device, and release of the lock responsive to control unit provided indicia; and

(b) executing by at least one control unit of a path expression inclusive of a RAID function including:

(1) requesting an explicit lock on at least one predetermined parity image from a counterpart device;
(2) inhibiting the execution of the RAID function until grant of the lock by the counterpart device;
(3) executing the RAID function on at least one data block and parity image of an associated parity set from the counterpart devices; and
(4) terminating the RAID function and providing indicia to the lock granting device.

14. In a storage subsystem having a plurality of cyclic, multitracked recording devices, each device storing blocks of data, a first and a second RAID control unit and coupling said recording devices, each control unit including an arrangement responsive to a write request for generating a parity block as a function of a set of data blocks and for writing the data blocks and the parity block for each set out to predetermined ones of the recording devices such that no single device stores more than one block from any one set and such that no single device stores all the parity blocks for the recorded sets, wherein the improvement at each device comprises:

a lock facility including a lock manager, a status list of lockable parity blocks and associated sets stored on the counterpart device, and a lock request queue;

first circuits including the lock facility responsive to a lock request from one or more control units for identifying at least the parity image of a parity set and either granting a lock to the first requesting device in enqueued order on a parity image if resident and available on the device or enqueueing the same;

second circuits including the lock facility for executing a sequence of accessing, updating, or regeneration tasks as specified by the control unit after grant of a lock; and
third circuits including the lock facility for releasing the lock responsive to termination indicia from the counterpart control unit.

15. An article of manufacture comprising a machine-readable memory having stored therein indicia of a plurality of processor-executable control program steps for path locking one of a plurality of RAID control units to at least one of a plurality of shareable cyclic, multitracked storage devices, the control units performing RAID functions on at least one data block and a parity image of a parity set defined over the devices, the blocks being distributed such that no single device stores more than one block from an associated data set, said plurality of indicia of control program steps include:

13

- (a) indicia of a control program step for defining a lock function on each device and causing said device to enqueue lock requests from accessing control units, grant lock requests to a requesting device in enqueued order to an available parity image stored on the device, and release of the lock responsive to control unit provided indicia; and
- (b) indicia of a control program step for executing a path expression inclusive of a RAID function by at least one control unit including:
- (1) requesting an explicit lock on at least one predetermined parity image from a counterpart device;
 - (2) inhibiting the execution of the RAID function until grant of the lock by the counterpart device;
 - (3) executing the RAID function on at least one data block and parity image of an associated parity set from the counterpart devices; and
 - (4) terminating the RAID function and providing indicia to the lock granting device.
16. A storage subsystem comprising:
- a plurality of storage devices having a lock function to lock parity image blocks in the storage device; and
- a first and a second control unit, each coupled to the plurality of storage devices, wherein to access a storage device, an accessing control unit executes a path expression, the path expression includes requesting a

14

lock from the storage device storing the parity image of the parity group addressed by the control unit, and enforcing a busy-wait until a lock is granted by the storage device, executing a RAID function, and then releasing the lock.

17. The storage subsystem of claim 16, wherein the storage devices have defined thereon one data block from a first parity set and a parity image of a second parity set, and wherein a storage device of the storage devices further comprises logic to enqueue any lock request embedded in a first read or write command embedded in a RAID function, said device being responsive to at least one of a sequence of commands from a control unit executing a path expression during the pendency by the device of any current lock.

18. The storage subsystem of claim 16, wherein executing a path expression includes forming a coroutines relationship with at least one of the storage devices.

19. The storage subsystem of claim 16, wherein the storage devices include logic to determine whether a parity set covered by a lock request is concurrently subject to a lock in whole or in part and if subject to a lock said lock request is enqueued, and if not subject to a lock said lock request is granted and any command associated with said request in the path expression is placed on a command queue for execution by the device.

* * * * *



US006449697B1

(12) **United States Patent**
Beardsley et al.

(10) Patent No.: **US 6,449,697 B1**

(45) Date of Patent: **Sep. 10, 2002**

(54) **PRESTAGING DATA INTO CACHE IN PREPARATION FOR DATA TRANSFER OPERATIONS**

5,802,566 A 9/1998 Hagersten

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

EP 0542483 A1 * 5/1993 G06F/12/10

OTHER PUBLICATIONS

"Programmable Support for Controlling Memory Sub-system Configurations in Personal Computers," IBM Technical Disclosure Bulletin, vol. 34, pp. 91-94, Mar. 1992.* ANSI, "Information Technology—Small Computer System Interface—2", reference No. X3.131-199X, revision 10L, Sep. 7, 1993, pp. 178-182.

IBM Technical Disclosure Bulletin, vol. 38, No. 6, "Methods of Specifying Data Prefetching Without Using a Separate Instruction", Jun. 1995, pp. 355-356.

IBM Technical Disclosure Bulletin, vol. 38, No. 9, "Method for 'Smart Prefetch' of Data from Main Memory by a Memory Controller for Access by a CPU", Sep. 1995, pp. 203-204.

Primary Examiner—Matthew Kim

Assistant Examiner—Pierre Michel Bataille

(74) Attorney, Agent, or Firm—David W. Victor, Konrad Raynes Victor & Mann LLP

(57) ABSTRACT

Disclosed is a method, system, and program for prestaging data into cache from a storage system in preparation for data transfer operations. A first processing unit communicates data transfer operations to a second processing unit that controls access to the storage system. The first processing unit determines addressable locations in the storage system of data to prestage into cache and generates a data structure capable of indicating contiguous and non-contiguous addressable locations in the storage system including the data to prestage into the cache. The first processing unit transmits a prestage command to the second processing unit. The prestage command causes the second processing unit to prestage into cache the data at the addressable locations indicated in the data structure. The first processing unit then requests data at the addressable locations indicated in the data structure. In response, the second processing unit returns the requested data from the cache.

48 Claims, 4 Drawing Sheets

(75) Inventors: **Brent Cameron Beardsley**, Tucson, AZ (US); **Jeffrey Allen Berger**, San Jose, CA (US)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/298,119**

(22) Filed: **Apr. 23, 1999**

(51) Int. Cl.⁷ **G06F 12/08**

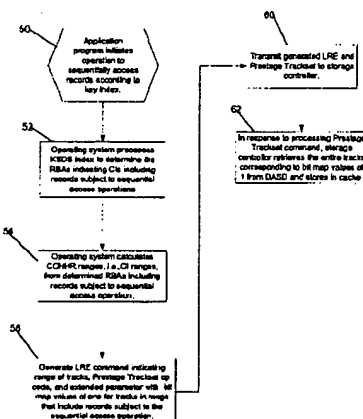
(52) U.S. Cl. **711/137; 711/113; 711/170; 710/36**

(58) Field of Search **711/137, 4, 112, 711/113, 114, 202, 138, 170, 171, 172, 209, 154; 710/36, 5**

(56) References Cited

U.S. PATENT DOCUMENTS

4,408,273 A 10/1983 Plow
4,855,907 A 8/1989 Ferro, Jr. et al.
4,980,823 A 12/1990 Liu
5,355,477 A 10/1994 Strickland et al.
5,377,345 A 12/1994 Chang et al.
5,426,761 A * 6/1995 Cord et al. 711/114
5,499,354 A 3/1996 Aschoff et al.
5,499,355 A 3/1996 Krishnamohan et al.
5,530,941 A 6/1996 Weisser et al.
5,596,736 A * 1/1997 Kerns 711/4
5,687,343 A * 11/1997 Fecteau et al. 711/202
5,721,865 A * 2/1998 Shintani et al. 712/207
5,724,548 A * 3/1998 Takahashi et al. 711/138
5,758,119 A 5/1998 Mayfield et al.
5,761,692 A * 6/1998 Ozden et al. 711/4
5,764,946 A 6/1998 Tran et al.
5,765,193 A * 6/1998 Rosich et al. 711/136



U.S. PATENT DOCUMENTS

5,809,529 A	9/1998	Mayfield		5,978,810 A	* 11/1999	Mitchell et al.	707/102
5,867,685 A	* 2/1999	Fuld et al.	711/113	6,185,659 B1	* 2/2001	Milillo et al.	711/137
5,943,689 A	* 8/1999	Tamer	711/166	6,260,115 B1	* 7/2001	Permut et al.	711/134

* cited by examiner

FIG. 1

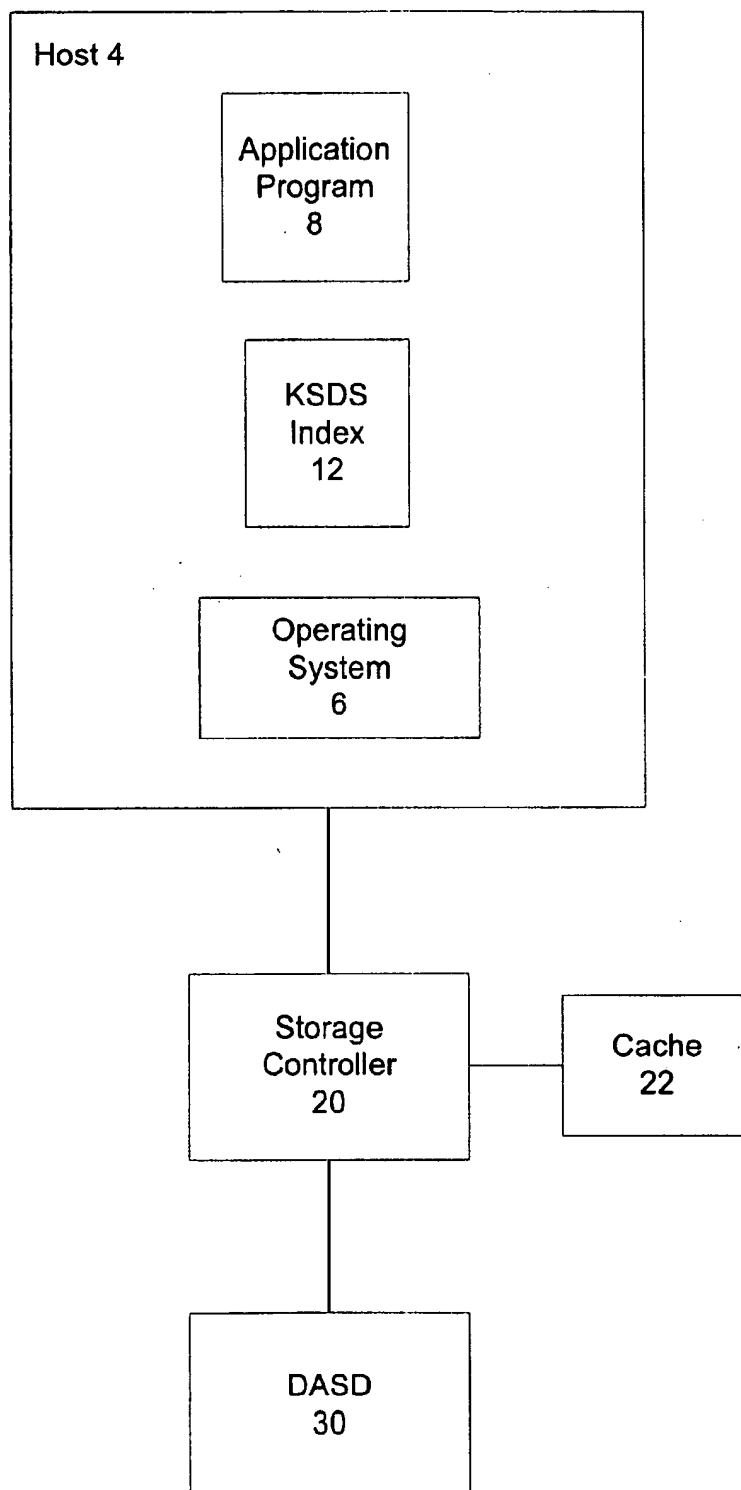


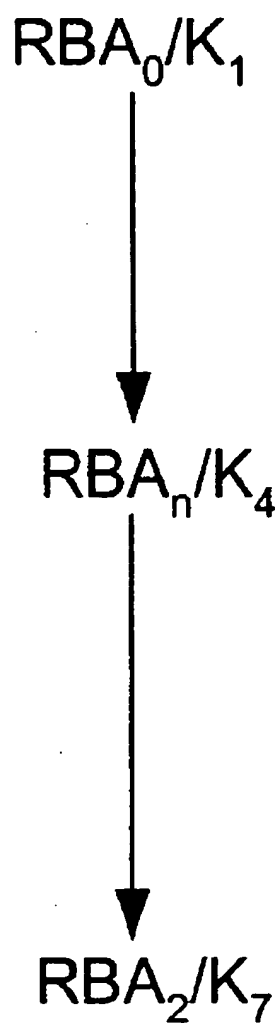
FIG. 2

FIG. 3

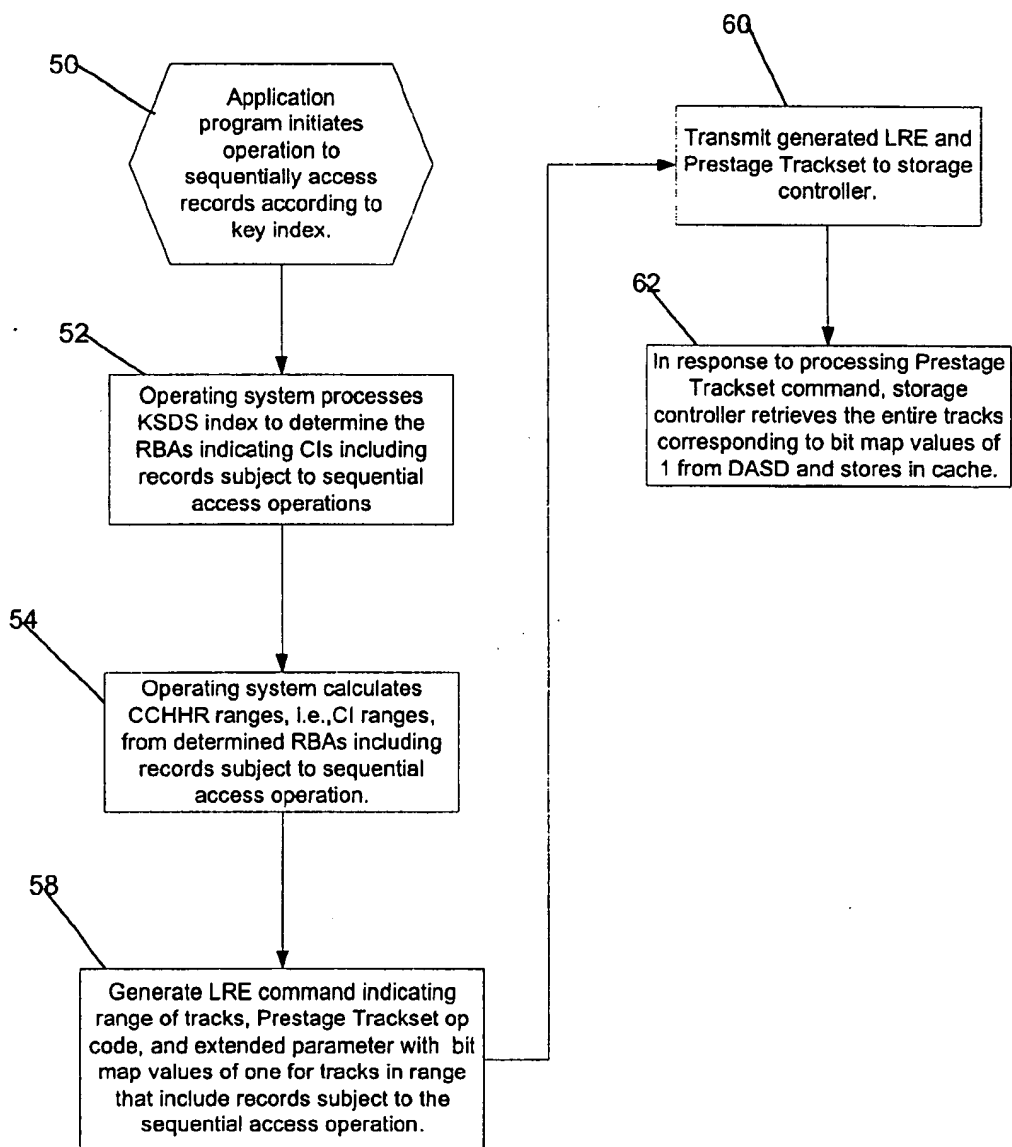
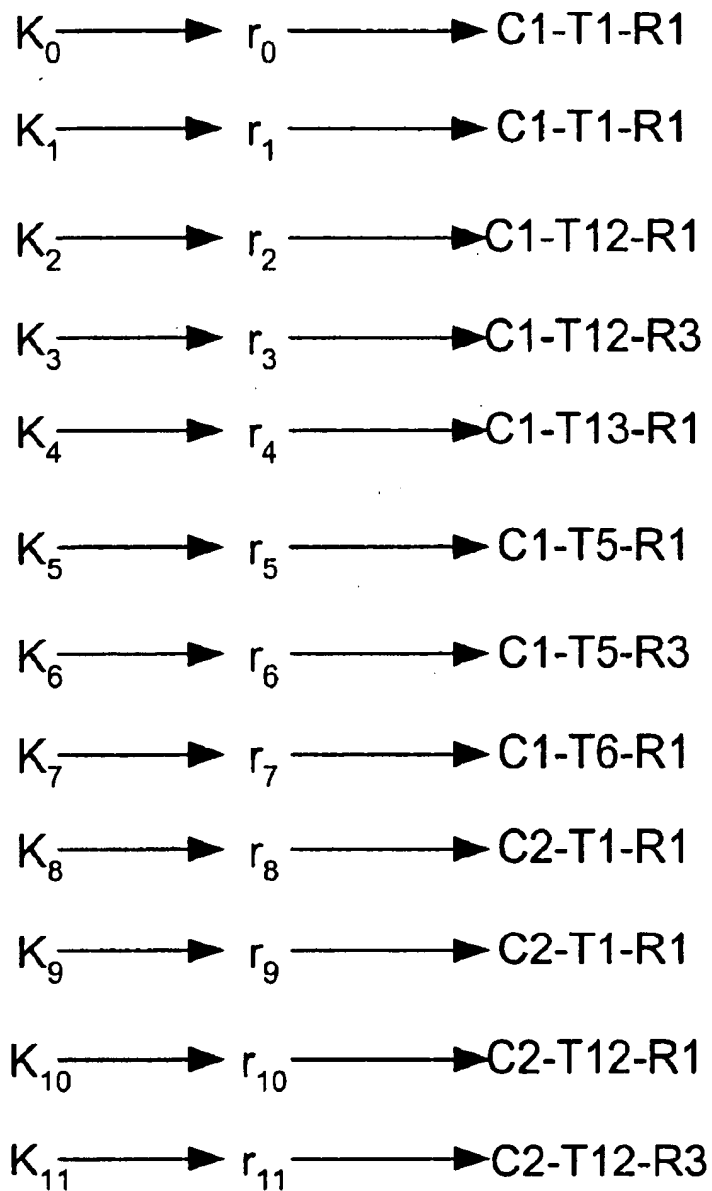


FIG. 4



1

PRESTAGING DATA INTO CACHE IN PREPARATION FOR DATA TRANSFER OPERATIONS

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to a method, system, and program for prestaging data into cache from a storage system in preparation for data transfer operations.

2. Description of the Related Art

Data prestaging techniques are used to prestage data from a non-volatile storage device, such as one or more hard disk drives, to a high speed memory, such as a volatile memory device referred to as a cache, in anticipation of future data requests. The data requests may then be serviced from the high speed cache instead of the storage device which takes longer to access. In this way, data may be returned to the requesting device faster.

During a sequential read operation, an application program, such as a batch program, will process numerous data records stored at contiguous locations in the storage device. It is desirable during such sequential read operations to prestage the sequential data into cache in anticipation of the requests from the application program. Present techniques used to prestage sequential blocks of data include sequential caching algorithms systems, such as those described in the commonly assigned patent entitled "CACHE DASD Sequential Staging and Method," having U.S. Pat. No. 5,426,761. A sequential caching algorithm detects when a device is requesting data as part of a sequential access operation. Upon making such a detection, the storage controller will begin prestaging sequential data records following the last requested data record into cache in anticipation of future sequential accesses. The cached records may then be returned to the application performing the sequential data operations at speeds substantially faster than retrieving the records from a non-volatile storage device.

Another prestaging technique includes specifying a block of contiguous data records to prestage into cache in anticipation of a sequential data request. For instance, the Small Computer System Interface (SCSI) provides a prefetch command, PRE-FETCH, that specifies a logical block address where the prestaging operation begins and a transfer length of contiguous logical blocks of data to transfer to cache. The SCSI PRE-FETCH command is described in the publication "Information Technology-Small Computer System Interface-2," published by ANSI on Apr. 19, 1996, reference no. X3.131-199x, Revision 10L, which publication is incorporated herein by reference in its entirety.

Both these techniques for prestaging data records in anticipation of sequential operations are not useful for data records that have a logical sequential relationship but are stored at non-contiguous or dispersed physical locations in the storage device. Such prior art prestaging techniques are intended for sequential operations accessing data records stored at contiguous physical locations. For instance, the sequential detection algorithms and SCSI PRE-FETCH command do not prestage non-contiguous blocks. If the sequential detection algorithms and the SCSI PRE-FETCH command are used to prestage a range of data records including both the non-contiguously stored data records that are needed, then they will also prestage data records that the application program does not need. The above techniques waste processor cycles and cache storage space by prestaging data records that will not be requested. Thus, current

2

prestaging techniques do not provide an optimal solution for prestaging non-contiguous tracks into cache.

Thus, there is a need in the art for improved prestaging techniques.

SUMMARY OF THE PREFERRED EMBODIMENTS

To overcome the limitations in the prior art described above, preferred embodiments disclose a method, system, and program for prestaging data into cache from a storage system in preparation for data transfer operations. A first processing unit communicates data transfer operations to a second processing unit that controls access to the storage system. The first processing unit determines addressable locations in the storage system of data to prestage into cache and generates a data structure capable of indicating contiguous and non-contiguous addressable locations in the storage system including the data to prestage into the cache. The first processing unit transmits a prestage command and the data structure to the second processing unit. The prestage command causes the second processing unit to prestage into cache the data at the addressable locations indicated in the data structure. The first processing unit then requests data at the addressable locations indicated in the data structure. In response, the second processing unit returns the requested data from the cache.

In alternative embodiments, the storage system storage space is logically divided into multiple tracks, wherein each track includes one or more data records. Each data record includes an index area providing index information on the content of the data record and a user data area including user data. The addressable locations indicated in the data structure comprise tracks in the storage system including the data records to prestage into the cache. In such embodiments, the data structure indicates addressable locations in the storage system including the data to prestage into the cache.

In yet further embodiments, the data structure comprises a bit map data structure having bit map values for addressable locations in the storage system. Bit map values of one in the data structure indicate corresponding addressable locations including the data to prestage into cache.

In still further embodiments, the addressable locations in the data structure correspond to data having a logical sequential ordering within the first processing unit.

In further instances, the addressable locations in the storage system including the data having the logical sequential ordering are at non-contiguous addressable locations in the storage system.

Preferred embodiments thus provide a mechanism to prestage data into cache using a data structure indicating addressable locations to prestage in a range of addressable locations. Preferred embodiments are particularly applicable to situations where an application program performs a sequential operation to process data records according to a logical sequential ordering. However, such data records having the logical sequential ordering may be stored at non-contiguous physical locations on a storage device. In such case, the data structure of the preferred embodiments can cause another processing unit, such as a storage controller that controls access to the storage system, to prestage into cache the data having a logical sequential relationship, yet stored at non-contiguous physical locations in the storage device. In this way, when the application program requests the data having the logical sequential ordering during sequential processing, the storage controller can return the requested data directly from cache. Returning the

data from cache is substantially faster than retrieving the requested data from non-contiguous physical locations from the storage device.

Preferred embodiments thus improve system performance for application programs performing sequential operations on data logically ordered yet stored at non-contiguous physical location by prestaging the logically ordered data into a high-speed cache.

BRIEF DESCRIPTION OF THE DRAWINGS

Referring now to the drawings in which like reference numbers represent corresponding parts throughout:

FIG. 1 is a block diagram illustrating a software and hardware environment in which preferred embodiments of the present invention are implemented;

FIG. 2 illustrates a block diagram of data structures utilized with preferred embodiments of the present invention;

FIG. 3 illustrates logic to prestage data in accordance with preferred embodiments of the present invention; and

FIG. 4 illustrates a mapping of key values, user data records, and physical storage locations that is utilized with preferred embodiments of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

In the following description, reference is made to the accompanying drawings which form a part hereof and which illustrate several embodiments of the present invention. It is understood that other embodiments may be utilized and structural and operational changes may be made without departing from the scope of the present invention.

Problems With Current Prestaging Techniques

A computer application program may maintain a logical mapping or index that maps data records to key values describing a particular ordering of data records. The computer may also maintain a physical mapping that maps the key values to physical locations on a storage device. There may also be additional mappings to map the data records to the exact physical locations on the storage device.

Oftentimes, an application program, such as batch programs, may want to sequentially access numerous data records according to key values that uniquely identify the data records. However, the actual data records, which are logically sequentially ordered according to the key values, may be dispersed throughout the storage device, i.e., not stored at contiguous physical locations. In such case, if the storage device is comprised of one or more hard disk drives or a tape storage device, then there will be latency delays while the storage device performs electro-mechanical set-up operations, e.g., moving an actuator arm to the read location, to access the non-contiguous physical storage locations on the disk drive storing the data records identified according to the logically sequential keys. Such latency delays in accessing non-contiguous physical locations could cause substantial delays in providing the application program with the records during the sequential processing of the logically sequential records.

The above described problem may arise when the International Business Machines Corporation (IBM) Virtual Storage Access Method (VSAM) is used to reference user data records stored in a storage device. The storage device may be a direct access storage device (DASD), which is comprised of numerous linked hard disk drives. A host computer

communicates I/O operations to a storage controller, such as the IBM 3990 Storage Controller, which controls access to the DASD. Physical locations on the DASD are identified according to cylinder, track, and the record on the track, i.e., CCHHR, where CC indicates the cylinder, HH indicates the track, and R indicates the record on the track.

One data set structure VSAM utilizes to store data is the Key Sequenced Data Sets (KSDS). KSDS is particularly applicable to data records that include a key value, which provides a unique identifier of the record. The storage space for a KSDS file is divided into a plurality of control areas (CAs), which may comprise a single cylinder of storage space or fifteen tracks. Each CA is comprised of multiple control intervals (CIs). A CI can be comprised of one or more records on the track. User data records are written to particular CIs. For instance, a CA may be comprised of 15 tracks, e.g., tracks 0-14, including one or more CIs for each track, and three free tracks, e.g., tracks 12, 13, and 14. The user may specify a certain amount of free space in each CI and a CA. The free space is provided for insertion and lengthening of records into a CI. The operating system would write user data records according to their logical key order to the first CI in a CA, i.e., CI₀, until the CIs in the CA were filled. With this system, data records would be written to the CIs according to their key ordering, e.g., the data record with the first key value as the first record in the first CI in the first CA. Thus, the data records would be written to the CIs according to the ordering of their keys at contiguous physical locations defined by the CIs.

One event that could cause user data records arranged according to the logical key value to be dispersed at non-contiguous physical locations in the storage device is CI and CA splitting. CI splitting occurs when an individual CI is filled with user data records. For instance, a CA may be set equivalent to a cylinder or 15 tracks, and tracks 0-11 are for user data and tracks 12-14 are specified as free. If user data records are written sequentially according to the key values to contiguous physical locations within the CIs in tracks 0-11 of the CA and a user data record needs to be inserted or lengthened, then the operating system will move half the user data records in the CI which is involved in the insert or lengthening operation to a CI in the free space of the CA, e.g., in track 12, to make room in the current CI for the inserted or lengthened record. With this control interval (CI) split, user data records, having a sequential logical key ordering, that were previously in the same CI, at contiguous physical locations, are now separated into non-contiguous CIs. The user data records moved to the free space CI in track 12 are no longer at contiguous physical locations with respect to the logically contiguous user data records that remained in the CI where the split occurred. For instance, if a CI had user records with key values of A, B, C, and D, and the C and D records were moved to track 12, then the C and D records would no longer be contiguous on the storage device to A and B. Hence, CI splitting causes user data records that are sequentially ordered with respect to logical index values to be stored at non-contiguous physical locations, e.g., non-contiguous CCHHRs.

Further, if all the tracks, including the free tracks, e.g., tracks 0-14, in a CA are filled with user data records, then the insertion or lengthening of a record in the CA, will cause a CA split, where certain records in the CA may have to be moved to a new CA to make room for the inserted or lengthened records. If the user data records were in a logical sequential order with respect to a key index, then such records moved to a different CA during the split would no longer have a sequential physical ordering as they are in a

different CA. Such splitting may occur whenever a record is inserted or an existing record is lengthened during sequential or direct processing of user data records.

Problems arise when the operating system and storage system fails to maintain sequential ordering in both the logical and physical domains. An application program may process records in a sequential mode with respect to a logical sequential ordering of key values. However, such logically sequential records, for the reasons discussed above, may not be stored at contiguous physical locations, i.e., contiguous CIs and CCHHRs. In such case, delays may occur in retrieving these logically sequential records stored at non-contiguous physical locations. If the application program is performing fast sequential operations, then delays in accessing the logically sequential data records at non-contiguous physical locations, which requires seek and rotation operations on the disk surface, may substantially degrade the performance of the application program's sequential operations.

Hardware and Software Environment

FIG. 1 illustrates a hardware and software environment in which preferred embodiments may be implemented. A host system 4 includes an operating system 6, an application program 8, and a KSDS index 12. The operating system 6 could comprise the IBM ESA/390 operating system including a data management component such as the IBM Distributed File Manager Storage Management System for MVS (DFSMS/MVS) software to provide the operating system 6 access to VSAM data sets and other data types. The application program 8 may comprise a batch program or a database program that performs sequential processing of data records according to a key ordering.

A storage controller 20 receives input/output (I/O) operations from the host system 4 and executes the received I/O operations against the direct access storage device (DASD) 30. A cache 22 is comprised of one or more volatile memory devices. The storage controller 20 would stage data tracks into the cache 22 that are retrieved from the DASD 30 in anticipation of subsequent requests for the data. Further, the storage controller 20 may prestage data tracks into cache before they are requested, in anticipation of such requests for the data. The DASD 30 may store data in a Count-Key-Data (CKD) format or fixed block format such as is used with SCSI. Details of a specific implementation of a storage controller 20, operating system 6, and data transfer operations there between are described in the IBM publications: "Enterprise Systems Architecture/390: ESCON I/O Interface," IBM document no. SA22-7202-02 (Copyright IBM Corp., 1990, 1991, 1992) and "IBM 3990 Storage Control Reference (Models 1, 2, and 3)," IBM document no. GA32-0099-06 (Copyright IBM Corp., 1998, 1994), which publications are incorporated herein by reference in their entirety.

In the CKD format, a count field provides the name and format of a record, the key length, and the data length. The key field, which is optional and unrelated to the index key used to provide logical ordering to the application program 8 records, is used for searching and may indicate the last data set in the record. The data field provides a variable length record of user data sets. The number of CKD records that can be placed on a track depends on the length of the data areas of the records. The physical location of a CKD record on a track is identified according to the R value of the CCHHR physical location identifier. The user data area of a CKD record may include multiple user data records, such as

data records (r) used by the application program 8 that correspond to key values (K). For instance, a track may include twelve 4K CKD records, i.e., four R values in a CCHH, and the user data area of each CKD record may be comprised of multiple application records (r) having corresponding key values (K).

The host system 4 may communicate with the storage controller 20 via channel paths by executing channel programs. In such case, data transfer operations are performed using channel command words (CCW) which are transferred from the host system 4 to the storage controller 20 along a channel path. The host system 4 may view the storage controller 18 as a multitude of separate control unit images or logical subsystems (LSSs), wherein each control unit image provides access to one or more I/O devices or LSS images of the DASD 30. Further details of how the host system 4 may communicate with the storage controller 20 are described in the commonly assigned and co-pending patent applications, all of which are incorporated herein by reference in their entirety: "Method And System For Dynamically Assigning Addresses To An Input/Output Device," by Brent C. Beardsley, Allan S. Merritt, Michael A. Paulsen, and Harry M. Yudenfriend, filed on Oct. 7, 1998, and having U.S. Pat. Ser. No. 09/167,782, U.S. Pat. No. 6,185,638; "System For Accessing An Input/Output Device Using Multiple Addresses," by Brent C. Beardsley, James L. Iskiyan, James McIlvain, Phillip R. Mills, Michael A. Paulsen, William G. Thompson, Harry M. Yudenfriend, filed on Oct. 7, 1998, and having U.S. Pat. Ser. No. 09/168,017, U.S. Pat. No. 6,170,023; "Method, System, and Program for Performing Data Transfer Operations on User Data," by Brent C. Beardsley and Michael A. Paulsen, filed on the same date hereof and having U.S. Pat. Ser. No. 09/298,154, U.S. Pat. No. 6,105,076.

The KSDS index 12 includes the key fields of the records and a relative byte address (RBA) that points to the CI that includes the user data record identified by the key field. The operating system 6 can calculate a CCHH location from the RBA as known in the art. The KSDS index 12 is organized in a tree structure. The entry node in the tree includes the high key value of the keys for the first sequential group of user records (r) that fit in the first CI and a RBA pointing to the physical CCHH location of the track including the first CI. When a CI is filled with user records (r), then the next group of user records, having key values sequential with respect to the records in the filled CI, will be placed in the next CI. The KSDS index 12 will then have a pointer from the first CI in the tree to the second CI that includes the next group of user records (r), sequential with respect to key values (K). This second index entry in the KSDS index 12 includes the high key value of those records in this second CI and the RBA of the second CI that includes such next sequential set of records.

FIG. 2 provides an example of the arrangement of the KSDS index 12. In this example, there are sequential keys K_0 - K_n , which are key values uniquely identifying user data records (r). The first CI₀ in the KSDS index 12 can store records K_0 and K_1 . Thus, the first entry in the KSDS index 12 would include the high key value of the user data records in the first CI₀ and the RBA₀ of that CI₀. In the example of FIG. 2, the next group of user data records are placed in the nth CI_n, which is not physically contiguous to the first CI. The CI_n may have to be used for the next sequential group of user data records, identified by keys K_2 and K_4 , if there is CI splitting. In the example of FIG. 2, CI_n can store the next three user data records corresponding to keys K_2 thru K_4 . The second KSDS index 12 entry to which the first index

entry points would thus include the high key value K_n and the RBA_n of CI_n , which is not contiguous to the first index entry RBA_0 . In the example of FIG. 2, CI_2 , which has a physical location contiguous to the first CI_1 , can store the next three data records, having keys K_3 thru K_7 . Thus, the third KSDS index 12 entry would include the high key value K_7 of the user data records stored in the CI_2 located at RBA_2 , which is at a physical contiguous location to the first CI_1 starting at the physical location specified by RBA_1 . In this way, the KSDS index 12 is logically contiguous with respect to key values, corresponding to user data records (r), but may not map those logically sequential data records, i.e., key values (K), to contiguous physical locations. Details on the VSAM implementation are described in IBM publication "VSE/VSAM User's Guide and Application Programming, Version 6 Release 1," IBM document no. SC33-6632-00 (Copyright IBM Corp. 1979, 1995) and the commonly assigned patent entitled "Method and Means for Cataloging Data Sets Using Dual Keyed Data Sets and Direct Pointers," U.S. Pat. No. 4,408,273, which publication and patent are incorporated herein by reference in their entirety.

Data Transfer Operations

The CCW format provides a format for a sequence of commands used to transfer data between the host system 4 and storage controller 20. The first command in the chain is a Define Extent command which defines the extent or range of tracks in which a channel program will operate. An extent is a set of consecutively addressed tracks that the channel program in the host 4 can access. The limits of an extent are defined by specifying the addresses of the first and last tracks in the extent. The Define Extent command further defines attributes of, and limitations on, the commands that follow in the channel program. Following is a Locate Record or Locate Record Extended command that specifies the operations, the number of consecutive records (or tracks) involved in the data operation, and the address of the first track and the orientation state to establish before starting data transfer. One or more read or write commands may follow the Locate Record command to perform data transfer operations. The storage controller 20 will perform the requested operation with respect to the DASD 30 and present status information to the host 4 indicating whether the operation failed or successfully completed.

Preferred embodiments include an additional command, referred to herein as a "Prestage Trackset" command, that is utilized within a CCW chain to provide notification to the storage controller 20 that a set of tracks will be accessed in a future operation. The Prestage Trackset command is included with a Locate Record Extended command and may be specified with a Prestage Trackset operation code within a Locate Record Extended parameter. If the Prestage Trackset operation code is specified, then the Locate Record Extended command would contain an Extended Parameter that provides a bit map of a range of tracks. Values in the bit map may be set to "0" or "1." A "1" value indicates that the track corresponding to the bit map value is to be prestaged into the cache 22, whereas a bit map value of "0" indicates that the corresponding track is skipped and not prestaged into cache 22. A Count parameter of the Locate Record Extended command indicates the number of tracks to be transferred with the Prestage Trackset operation. The Count parameter is equal to the number of bit map values of one in the Extended Parameter bit map, i.e., those tracks in the range of sequential tracks to be prestaged.

The first bit in the bit map must be '1' and represents the track whose address is specified in the seek address param-

eter. Subsequent addressed tracks are in ascending order. In preferred embodiments, tracks in the bit map represented by one bits are not limited to the tracks contained within the extent domain defined in the Define Extent.

In preferred embodiments, a single CCW chain may include a Prestage Trackset command and data transfer commands. The Prestage Trackset command would prestage data in anticipation of read requests for such data in subsequent CCW chains. The read operations in the CCW chain including the Prestage Trackset should be for data tracks that were prestaged into cache in a previous CCW chain. In this way, in a single CCW chain, tracks that will be needed in future operations can be prestaged into cache and tracks previously prestaged can be read from cache.

Data transfer operations in the CCW chain including the Prestage Trackset command would follow a Locate Record or Locate Record Extended command specifying such data transfer operations. This subsequent Locate Record or Locate Record Extended command would follow the Locate Record Extended Command including the Prestage Trackset operation. Thus, in preferred embodiments, the data transfer operations occur in a Locate Record domain following the execution and completion of the Prestage Trackset operation. This insures that tracks are prestaged before any subsequent data transfer operations are performed. Further, in preferred embodiments, the Extended Parameter bit map used by the Prestage Trackset command may specify tracks to prestage that are not within the domain specified in the Define Extent command beginning the CCW chain including the Prestage Trackset operation. In a chain including both the Prestage Trackset operation and data transfer commands, the Define Extent domain may specify the range in which data operations are performed and Prestage Trackset operations may fall outside of this domain. In further embodiments, Prestage Trackset operations and read operations can occur in any order in a CCW chain.

FIG. 3 illustrates program logic to generate and process the Prestage Trackset command within both the host 4 and storage controller 20 systems. The operations shown in FIG. 3 as implemented in the host 4 and storage controller 20 may be executed asynchronously. Control begins at block 50 where the application program 8 initiates an operation to sequentially access numerous user data records according to a key index ordering assigned by the application program 8. The application program 8 maintains information on the logical sequential arrangement of user data records according to key (K) values, as well as information about free space; such information is unknown to the storage controller 20. The operating system 6 would then process (at block 52) the KSDS index 12 to determine the RBAs pointing to CIs including the records (r) subject to the sequential access operations. The operating system 6 then calculates (at block 54) the CCHHR ranges, i.e., CI ranges, from the determined RBAs including records subject to sequential access operations. The CCHHR range of the CI can be determined from the RBA as the RBA indicates the starting CCHHR of the CI and the CI has a fixed length which is used to determine the ending CCHHR of the CI. As discussed, these CCHHR ranges, i.e., CIs, including the logically sequential user records (r) may be at non-contiguous physical locations.

The operating system 6 would then generate (at block 58) a CCW chain including a Locate Record Extended command indicating a range of tracks, a Prestage Trackset operation code, and an Extended Parameter bit map indicating, with a bit map value of one, all tracks within the range of tracks that include user data records subject to the application's 8 sequential operation. For tracks that contain

nothing but free space, corresponding bits in the Prestage Trackset bit map will always be zero. The operating system 6 would then transfer (at block 60) the CCW chain to the storage controller 8. In preferred embodiments, the CCW chain including the Prestage Trackset operation may include an additional Locate Record domain with read commands. In response to receiving the Prestage Trackset command, the storage controller 20 would retrieve (at block 62) the data on the tracks having a corresponding bit map value of one in the Extended Parameter bit map. In preferred embodiments, the Prestage Trackset command causes the storage controller 20 to transfer the entire contents of a track, including the count and key data, into the cache 22. After prestaging the data into cache 22, the storage controller 20 would return a channel end and device end status. Following the prestage operation, the host 4 may then, within the same CCW chain, transfer data transfer operations.

In this way, with the Prestage Trackset command, the host 4 can have data records that may be stored at non-contiguous physical locations staged into cache 22 in anticipation that a host 4 application program, such as the application program 8, is sequentially accessing such data records according to a logical arrangement of data, e.g., according to key.

The Prestage Trackset command of the preferred embodiments may be utilized with the Read Track Data command described in the co-pending and commonly assigned patent application entitled, "Method, System, and Program for Performing Data Transfer Operations on User Data," by Brent C. Beardsley and Michael A. Paulsen, filed on the same date hereof and having U.S. Pat. No. 6,105,067, incorporated by reference above. The Read Track Data command requests the storage controller 20 to transfer to the host 4 all the user data records on a track, following the first user data record, R_0 , free of any of the count and key data on the track.

FIG. 4 illustrates how user data records r_0-r_{11} , which are organized sequentially with respect to keys K_0-K_{11} , map to non-contiguous physical locations, e.g., CCHHR locations, in the DASD 30. As discussed, the data records (r) are not stored sequentially to match the logical sequential key ordering as a result of CI and CA splitting. For example, r_0 , which is the first record in the key ordering, is in the CI starting at cylinder 1 (C1), track 1 (T1), and the first CKD record (R1). The next record r_1 in the logical sequential ordering, having key value K_2 , is stored in the same CI as r_0 . User record r_2 is stored in track 12. Thus, the logically sequential user records r_0-r_2 are stored at non-contiguous physical locations.

Below is pseudo code for three CCW chains generated by the host 4 to prestage data before and during the application program 8 sequentially accessing the records corresponding to the first twelve key values, K_0 thru K_{11} , as shown in FIG. 4. A CCW chain may use the Prestage Trackset command in combination with one or more Read Track Data Commands. Two Locate Record Extended (LRE) commands may be included in a CCW chain, one for the Prestage Trackset command and the other for a string of Read Track Data commands. The Locate Record Extended command for the Prestage Trackset operation would prestage tracks having a bit map value of one in the Extended Parameter bit map. A second Locate Record Extended command may follow the Locate Record Extended command for the prestage trackset operation. This second Locate Record Extended command may specify an Extended Parameter bit map of non-sequential tracks in a range of tracks subject to a sequence of Read Track Data commands to read user data records from the tracks having a bit map value of one. In preferred

embodiments, the host 4 would transfer the read commands and accompanying Locate Record command after receiving channel end and device end status indicating completion of the prestage operation.

Below are three exemplar operations to prestage user data records and transfer user data records while the application program 8 is sequentially accessing and processing records r_0 thru r_{11} , ordered sequentially with respect to key values K_0 thru K_{11} , as shown in FIG. 4.

Operation 1

Define Extent (C1-T1 to C1-T12);

Locate Record Extended (Prestage Trackset, Bit map of twelve values representing tracks 1-12, with bit map values of 1 for tracks 1 and 12);

Operation 2:

Define Extent (C1-T1 to C1-T12);

Locate Record Extended (Prestage Trackset, Bit map of 13 values representing tracks 1-13, with bit map values for tracks 5, 6, and 13 set to 1) Locate Record Extended (Read Track op code, Read Trackset with bit map of twelve values with bit map values for tracks 1 and 12 set to 1);

Read Track Data from track 1;

Read Track Data from track 12;

Operation 3:

Define Extent (Range C1-T5 to C1-T13) Locate Record Extended (Prestage Trackset, Bit map with twelve values with bit map values for tracks 1 and 12 in cylinder 2 set to 1);

Locate Record Extended (Read Track op code, Read Trackset with bit map of nine values with bit map values for tracks 5, 6, and 13 in cylinder 1 set to 1);

Read Track Data from track 5 in cylinder 1;

Read Track Data from track 6 in cylinder 1;

Read Track Data from track 13 in cylinder 1;

Operation 4:

Define Extent (Range C2-T1 to C2-T12) Locate Record Extended (Read Track op code, Read Trackset with bit map of twelve values with bit map values for tracks 1 and 12 in cylinder 2 set to 1);

Read Track Data from track 1 in cylinder 2;

Read Track Data from track 12 in cylinder 2;

In operation 1, the host 4 is performing set-up operations to prestage the tracks including the first four records to be sequentially accessed, r_0 to r_3 by the application program 8. As discussed there may be multiple user records (r) in a CKD record identified according to CCHHR. The host 4 sends a Prestage Trackset command to the storage controller 20 to prestage tracks 1 and 12 from cylinder 1 into the cache 22. The parameters in the Define Extent command refer to the start and end of the extent in which the following operations will be performed.

After prestaging the first set of records, r_0 to r_3 , the host 4 will issue a CCW operation 2 including a Prestage Trackset operation to prestage the tracks including the next four records, r_4 to r_7 , to be sequentially accessed. The Define Extent command specifies a range of tracks, tracks 1-12, that include the tracks that will be read in the CCW chain of operation 2, and does not include the tracks to prestage that are outside of the domain of the tracks to read. For this reason, the Define Extent command in Operation 2 does not specify tracks to prestage, such as track 13, that are outside of the domain including the tracks to read. Following is a Locate Record Extended command including a bit map indicating the tracks T1 and T12 that include the first four records, R_0 to R_3 , that the application program 8 will access. Following is a sequence of Read Track Data commands to

read the all user data records, free of any count or key information, from tracks 1 and 12 including user data records R_0 to R_3 which were prestaged into cache 22 in the CCW chain of operation 1. The sequence of Read Track Data commands apply to the tracks indicated in the Read Trackset bit map having a corresponding bit map value of one. Alternatively, the host 4 may use a sequence of Read Data commands to read a specific user data record from the tracks previously prestaged into cache 22, instead of using the Read Track Data command to read all the records from a track previously prestaged into cache 22. In both cases, because the requested data was prestaged into cache 22, the storage controller 20 may return the data from cache 22.

After operation 2, the application program 8 may begin performing operations on the first four sequential user data records R_0 to R_3 after the records are transferred from the cache 22 to the host 4. In operation 3, as the application program 8 is processing the first four records R_0 to R_3 as part of a sequential operation, the host 4 performs operation 3 by issuing a Locate Record Extended command indicating the Prestage Trackset operation to cause the storage controller 20 to prestage from the DASD 30 into cache 22 those tracks T1 and T12 in cylinder 2 including the user data records R_8 to R_{11} . The host 4 would specify in the Extended Parameter bit map that tracks 1 and 12 on cylinder 2 are to be prestaged into cache 22. The second Locate Extended Record command indicates a bit map indicating that tracks 5, 6, and 13 are subject to a series of read operations to read the user data records from the CKD records in tracks 5, 6, and 13, which were previously prestaged into cache 22 in operation 2. The read operation may comprise a Read Track Data command or Read Data command to read specific user data records prestaged into cache 22.

After performing operation 3, the application program 8 may then process user data records R_4 to R_7 . While processing these records, the host 4 may then perform operation 4 to retrieve the final set of logically sequential records R_8 to R_{11} that the application program 8 will process. Operation 4 consists of a Locate Record Extended command indicating a read operation code, for a Read Track Data command or Read Data command, to be performed on the tracks corresponding to bit map values of one in the Extended Parameter bit map provided with the Locate Record Extended command.

With the above four operations, the host 4 may cause the storage controller 20 to prestage non-contiguous tracks into cache 22 in anticipation that the host 4 will later request user data records from these tracks for a sequential operation performed on user data records arranged in a logical sequence according to a key or other index type value. In this way, the host 4 can retrieve user data records the application program 8 needs without having to wait for the storage controller 20 to perform mechanical set-up operations to retrieve the data from DASD 30, e.g., track and seek movements to position a read head or tape set-up operations. Instead, the storage controller 20 may return the requested data directly from cache 22.

Preferred embodiments thus allow an application performing a sequential processing operation on user data records according to a logical sequential ordering to immediately access the user data records that are in the logical sequential order, but stored at non-contiguous physical locations. This provides an operation sequentially accessing records according to a logical key ordering that are stored at non-contiguous physical locations with the same performance that is achieved when performing a sequential operation on data that is stored at contiguous physical locations.

Conclusion

This concludes the description of the preferred embodiments of the invention. The following describes some alternative embodiments for accomplishing the present invention.

The preferred embodiments may be implemented as a method, apparatus or article of manufacture using standard programming and/or engineering techniques to produce software, firmware, hardware, or any combination thereof. The term "article of manufacture" (or alternatively, "computer program product") as used herein is intended to encompass one or more computer programs and data files accessible from one or more computer-readable devices, carriers, or media, such as a magnetic storage media, "floppy disk," CD-ROM, a file server providing access to the programs via a network transmission line, holographic unit, etc. Of course, those skilled in the art will recognize many modifications may be made to this configuration without departing from the scope of the present invention.

Preferred embodiments were described with respect to sequential data transfer operations which involve reading or writing numerous, user data records in a logical sequential relationship. However, in alternative embodiments, the preferred embodiment commands, such as the Prestage Trackset command, may be used to prestage tracks in preparation for non-sequential, direct processing or random data transfer operations.

Preferred embodiments provided specific naming conventions for the data transfer operations described herein, such as Prestage Trackset, Read Data, Read Track Data, etc. However, any naming scheme or format may be used in implementing the commands which perform the functions described herein.

Preferred embodiments were described with respect to a storage controller, host, and DASD system. In alternative embodiments, the preferred embodiment commands may be used with any type of storage system arrangement, where one processing unit performs data operations with respect to a storage device by communicating with another processing unit that manages and controls access to the storage device. The storage device storing the data may be any storage device known in the art, including volatile and non-volatile storage devices, such as tape drives, flash memory, optical disk drives, etc. For instance, the commands may be used with any processor-to-processor communication, regardless of the environment in which the processors are implemented, i.e., the same computer, a network environment, etc. Further the cache 22 may be any type of volatile or non-volatile storage area utilized by the storage controller 20 for data transfer operations.

Preferred embodiments were described as implemented with certain operating system and application programs. However, these components were described as one preferred implementation. The preferred embodiment commands may be utilized in any operating system environment and with any application program which sequentially processes user data records maintained in a logical sequential ordering. Preferred embodiments are particularly applicable to situations where the application program wants the performance of a sequential type operation on the logically arranged records.

Preferred embodiments were described with respect to the CKD record format, where user data is stored in CKD records on a track, such that each CKD record includes a count field and may include a key field. The preferred embodiment commands may apply to other data storage

formats in which data is stored in records that include index information, such as the count and/or key type information, along with the user data. Further preferred embodiments may apply to the SCSI storage format which stores data in fixed blocks without the use of index information with each data record. The prestaging methods of the preferred embodiments may further be used with the partition data set extended (PDSE) storage format in which records are stored as fixed block records.

In the SCSI or PDSE formats, the preferred embodiment commands may be used to prestage data stored at non-sequential fixed block addresses on the storage device to a cache in anticipation of sequential data operations performed on data records physically stored at non-sequential fixed block addresses on the storage device. Thus, those skilled in the art will appreciate that the preferred embodiment commands may apply to any data storage format where data records maintained in a logical sequential ordering may nonetheless be stored at non-sequential physical locations on the storage medium.

Preferred embodiments were described with respect to a KSDS index used to map logically sequential records to relative byte address (RBAs) that are converted to CCHHR locations. However, the index used to map the logically sequential records to physical locations may comprise any such mapping and indexing technique known in the art. For instance, if the data format is SCSI, then the index may include entries indicating fixed block addresses as the starting address of a range of logically sequential records, as opposed to the RBA value that indicates a CCHHR location.

Preferred embodiments were described with respect to a bit map data structure that indicated tracks to prestage. In alternative embodiments, the unit to prestage may be different than a track, such as a one or more records, fixed blocks, etc. Still further, data structures other than a bit map may be used to indicate the data unit to prestage into cache.

In summary, preferred embodiments disclose a method, system, and program for prestaging data into cache from a storage system in preparation for data transfer operations. A first processing unit communicates data transfer operations to a second processing unit that controls access to the storage system. The first processing unit determines addressable locations in the storage system of data to prestage into cache and generates a data structure capable of indicating contiguous and non-contiguous addressable locations addressable locations in the storage system including the data to prestage into the cache. The first processing unit transmits a prestage command to the second processing unit. The prestage command causes the second processing unit to prestage into cache the data at the addressable locations indicated in the data structure. The first processing unit then requests data at the addressable locations indicated in the data structure. In response, the second processing unit returns the requested data from the cache.

The foregoing description of the preferred embodiments of the invention has been presented for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching. It is intended that the scope of the invention be limited not by this detailed description, but rather by the claims appended hereto. The above specification, examples and data provide a complete description of the manufacture and use of the composition of the invention. Since many embodiments of the invention can be made without departing from the spirit and scope of the invention, the invention resides in the claims hereinafter appended.

What is claimed is:

1. A method for prestaging data into cache from a storage system in preparation for data transfer operations, wherein a first processing unit communicates data transfer operations to a second processing unit that controls access to the storage system, comprising the first processing unit:

- (a) determining addressable locations in the storage system of data to prestage into cache;
- (b) generating a data structure capable of indicating contiguous and non-contiguous addressable locations in the storage system including the data to prestage into the cache by:
 - (i) generating a bit map data structure having bit map values for addressable locations in the storage system; and
 - (ii) setting to one the bit map values corresponding to the addressable locations including the data to prestage into cache;
- (c) transmitting a prestage command to the second processing unit which controls access to the storage system, wherein the prestage command causes the second processing unit to prestage into cache the data at the addressable locations indicated in the data structure; and
- (d) requesting data at the addressable locations indicated in the data structure, wherein the second processing unit returns the requested data from the cache.

2. The method of claim 1, wherein determining the addressable locations to prestage, comprises:

- determining, with the first processing unit, data having a logical sequential ordering; and
- determining, with the first processing unit, the addressable locations in the storage system of the data having the logical sequential ordering, wherein the determined addressable locations include the data to prestage into the cache.

3. The method of claim 2, wherein the addressable locations in the storage system including the data having the logical sequential ordering are at non-contiguous addressable locations in the storage system.

4. The method of claim 2, wherein the determination of the addressable locations of the data having the logical sequential ordering is determined from a Volume Storage Access Method (VSAM) Key Sequenced Data Set (KSDS) index.

5. The method of claim 1, wherein the storage system storage space is logically divided into multiple tracks, wherein each track, includes one or more data records, wherein each data record includes an index area providing index information on the content of the data record and a user data area including user data, wherein the addressable locations indicated in the data structure comprise tracks in the storage system including the data records to prestage into the cache.

6. The method of claim 1, wherein the requested data that the second processing unit returns from cache was prestaged into cache in a command sequence preceding the command sequence including the data request.

7. A method for prestaging data into cache from a storage system in preparation for data transfer operations, wherein a first processing unit communicates data transfer operations to a second processing unit that controls access to the storage system, comprising the second processing unit:

- receiving a prestage command from the first processing unit and a data structure capable of indicating contiguous and non-contiguous addressable locations in the

15

storage system including the data to prestage into the cache, wherein the data structure comprises a bit map data structure having bit map values for addressable locations in the storage system, wherein the bit map values corresponding to the addressable locations including the data to prestage into cache are set to one; prestaging into the cache the data at the addressable locations indicated in the data structure; receiving a data request from the first processing unit for data at the addressable locations indicated in the data structure; and returning to the first processing unit the requested data from the cache.

8. The method of claim 7, wherein the data structure comprises a bit map data structure having bit map values for addressable locations in the storage system, wherein the second processing unit prestages into cache data at the addressable locations having a corresponding bit map value of one.

9. The method of claim 7, wherein the addressable locations in the data structure correspond to data having a logical sequential ordering within the first processing unit.

10. The method of claim 9, wherein the addressable locations in the storage system including the data having the logical sequential ordering are at non-contiguous addressable locations in the storage system.

11. The method of claim 7, wherein the storage system storage space is logically divided into multiple tracks, wherein each track includes one or more data records, wherein each data record includes an index area providing index information on the content of the data record and a user data area including user data, wherein the addressable locations indicated in the data structure comprise tracks in the storage system including the data records to prestage into the cache.

12. The method of claim 7, wherein the data request from the first processing unit is for data that was prestaged into cache in a command sequence preceding the command sequence including the data request.

13. A method for prestaging data into cache from a storage system in preparation for data transfer operations, wherein a first processing unit communicates data transfer operations to a second processing unit that controls access to the storage system, wherein the storage system storage space is logically divided into multiple tracks, wherein each track includes one or more data records, wherein each data record includes an index area providing index information on the content of the data record and a user data area including user data, comprising the first processing unit:

determining addressable locations in the storage system of data to prestage into cache;

generating a data structure capable of indicating contiguous and non-contiguous addressable locations in the storage system including the tracks to prestage into the cache by:

(i) generating a bit map data structure having bit map values for addressable locations in the storage system; and

(ii) setting to one the bit map values corresponding to the addressable locations including the tracks to prestage into cache;

transmitting a prestage command to the second processing unit which controls access to the storage system, wherein the prestage command causes the second processing unit to prestage into cache the tracks at the addressable locations indicated in the data structure; and

16

requesting data at the addressable locations indicated in the data structure, wherein the second processing unit returns the requested data from the cache.

14. A method for prestaging data into cache from a storage system in preparation for data transfer operations, wherein a first processing unit communicates data transfer operations to a second processing unit that controls access to the storage system, wherein the storage system storage space is logically divided into multiple tracks, wherein each track includes one or more data records, wherein each data record includes an index area providing index information on the content of the data record and a user data area including user data, comprising the second processing unit:

receiving a prestage command from the first processing unit and a data structure capable of indicating a contiguous and non-contiguous range of addressable locations in the storage system including the tracks to prestage into the cache, wherein the data structure comprises a bit map data structure having bit map values for addressable locations in the storage system, wherein the bit map values corresponding to the addressable locations including the tracks to prestage into cache are set to one;

prestaging into the cache the tracks at the addressable locations indicated in the data structure;

receiving a data request from the first processing unit for data at the addressable locations indicated in the data structure; and

returning to the first processing unit the requested data from the cache.

15. A system for transferring commands to a controller to prestage data into a cache from a storage system controlled by the controller in preparation for data transfer operations, comprising:

a processing unit;

program logic executed by the processing unit, comprising:

(i) means for determining addressable locations in the storage system of data to prestage into cache;

(ii) means for generating a data structure capable of indicating contiguous and non-contiguous addressable locations in the storage system including the data to prestage into the cache by:

(a) generating a bit map data structure having bit map values for addressable locations in the storage system; and

(b) setting to one the bit map values corresponding to the addressable locations including the data to prestage into cache;

(iii) means for transmitting a prestage command to the controller, wherein the prestage command causes the controller to prestage into cache the data at the addressable locations indicated in the data structure; and

(iv) means for requesting data at the addressable locations indicated in the data structure from the controller, wherein the controller returns the requested data from the cache.

16. The system of claim 15, wherein the program logic for determining the addressable locations to prestage, further comprises:

means for determining data having a logical sequential ordering; and

means for determining the addressable locations in the storage system of the data having the logical sequential ordering, wherein the determined addressable locations include the data to prestage into the cache.

17

17. The system of claim 16, wherein the addressable locations in the storage system including the data having the logical sequential ordering are at non-contiguous addressable locations in the storage system.

18. The system of claim 16, wherein the determination of the addressable locations of the data having the logical sequential ordering is determined from a Volume Storage Access Method (VSAM) Key Sequenced Data Set (KSDS) index.

19. The system of claim 15, wherein the storage system storage space is logically divided into multiple tracks, wherein each track includes one or more data records, wherein each data record includes an index area providing index information on the content of the data record and a user data area including user data, wherein the addressable locations indicated in the data structure comprise tracks in the storage system including the data records to prestage into the cache.

20. The system of claim 15, wherein the requested data is for data that was prestaged into cache in a command sequence preceding the command sequence including the data request.

21. The system of claim 15, wherein data in the storage system is in a count-key-data format, and wherein the system comprises a host computer system and the controller comprises a storage controller, and the storage system comprises a direct access storage device (DASD).

22. A controller for prestaging data in preparation for data transfer operations from a computer system, wherein the controller controls access to a storage system, comprising:

a processing unit;

a cache accessible to the processing unit;

program logic executed by the processing unit, comprising:

- (i) receiving a prestage command from the computer system and a data structure capable of indicating contiguous and non-contiguous addressable locations in the storage system including the data to prestage into the cache, wherein the data structure comprises a bit map data structure having bit map values for addressable locations in the storage system, wherein the bit map values corresponding to the addressable locations including the data to prestage into cache are set to one;
- (ii) prestaging into the cache the data at the addressable locations indicated in the data structure;
- (iii) receiving a data request from the computer system for data at the addressable locations indicated in the data structure; and
- (iv) returning to the computer system the requested data from the cache.

23. The controller of claim 22, wherein the data structure comprises a bit map data structure having bit map values for addressable locations in the storage system, wherein the controller prestages into cache data at the addressable locations having a corresponding bit map value of one.

24. The controller of claim 22, wherein the addressable locations in the data structure correspond to data having a logical sequential ordering within the computer system.

25. The controller of claim 24, wherein the addressable locations in the storage system including the data having the logical sequential ordering are at non-contiguous addressable locations in the storage system.

26. The controller of claim 22, wherein the storage system storage space is logically divided into multiple tracks, wherein each track includes one or more data records, wherein each data record includes an index area providing

18

index information on the content of the data record and a user data area including user data, wherein the addressable locations indicated in the data structure comprise tracks in the storage system including the data records to prestage into the cache.

27. The controller of claim 22, wherein the data request from the computer system is for data that was prestaged into cache in a command sequence preceding the command sequence including the data request.

28. The controller of claim 22, wherein data in the storage system is in a count-key-data format, and wherein the computer system comprises a host computer system and the controller comprises a storage controller, and the storage system comprises a direct access storage device (DASD).

29. A system for transferring commands to a controller to prestage data into a cache from a storage system controlled by the controller in preparation for data transfer operations, wherein the storage system storage space is logically divided into multiple tracks, wherein each track includes one or more data records, wherein each data record includes an index area providing index information on the content of the data record and a user data area including user data, comprising:

a processing unit;

program logic executed by the processing unit, comprising:

- (i) means for determining addressable locations in the storage system of data to prestage into cache;
- (ii) means for generating a data structure capable of indicating contiguous and non-contiguous addressable locations in the storage system including the tracks to prestage into the cache by:
 - (a) generating a bit map data structure having bit map values for addressable locations in the storage system; and
 - (b) setting to one the bit map values corresponding to the addressable locations including the tracks to prestage into cache;
- (iii) means for transmitting a prestage command to the controller, wherein the prestage command causes the controller to prestage into cache the tracks at the addressable locations indicated in the data structure; and
- (iv) means for requesting data at the addressable locations indicated in the data structure from the controller, wherein the controller returns the requested data from the cache.

30. A controller for prestaging data in preparation for data transfer operations from a computer system, wherein the controller controls access to a storage system, wherein the storage system storage space is logically divided into multiple tracks, wherein each track includes one or more data records, wherein each data record includes an index area providing index information on the content of the data record and a user data area including user data, comprising:

a processing unit;

a cache accessible to the processing unit;

program logic executed by the processing unit, comprising:

- (i) receiving a prestage command from the computer system and a data structure capable of indicating contiguous and non-contiguous addressable locations in the storage system including the data to prestage into the cache, wherein the data structure comprises a bit map data structure having bit map values for addressable locations in the storage

system, wherein the bit map values corresponding to the addressable locations including the data to prestage into cache are set to one;

- (ii) prestaging into the cache the tracks at the addressable locations indicated in the data structure;
- (iii) receiving a data request from the computer system for data at the addressable locations indicated in the data structure; and
- (iv) returning to the computer system unit the requested data from the cache.

31. An article of manufacture including prestage commands to prestage data into cache from a storage system in preparation for data transfer operations, wherein a first processing unit communicates data transfer operations to a second processing unit that controls access to the storage system, the article of manufacture comprising computer readable storage media including at least one computer program embedded therein that is capable of causing the first processing unit to perform:

- (a) determining addressable locations in the storage system of data to prestage into cache;
- (b) generating a data structure capable of indicating contiguous and non-contiguous addressable locations in the storage system including the data to prestage into the cache by:
 - (i) generating a bit map data structure having bit map values for addressable locations in the storage system; and
 - (ii) setting to one the bit map values corresponding to the addressable locations including the data to prestage into cache;
- (c) transmitting a prestage command to the second processing unit which controls access to the storage system, wherein the prestage command causes the second processing unit to prestage into cache the data at the addressable locations indicated in the data structure; and
- (d) requesting data at the addressable locations indicated in the data structure, wherein the second processing unit returns the requested data from the cache.

32. The article of manufacture of claim 31, wherein determining the addressable locations to prestage, comprises:

- determining, with the first processing unit, data having a logical sequential ordering; and
- determining, with the first processing unit, the addressable locations in the storage system of the data having the logical sequential ordering, wherein the determined addressable locations include the data to prestage into the cache.

33. The article of manufacture of claim 32, wherein the determination of the addressable locations of the data having the logical sequential ordering is determined from a Volume Storage Access Method (VSAM) Key Sequenced Data Set (KSDS) index.

34. The article of manufacture of claim 32, wherein the addressable locations in the storage system including the data having the logical sequential ordering are at non-contiguous addressable locations in the storage system.

35. The article of manufacture of claim 31, wherein the storage system storage space is logically divided into multiple tracks, wherein each track includes one or more data records, wherein each data record includes an index area providing index information on the content of the data record and a user data area including user data, wherein the addressable locations indicated in the data structure com-

prise tracks in the storage system including the data records to prestage into the cache.

36. The article of manufacture of claim 31, wherein the requested data that the second processing unit returns from cache was prestaged into cache in a command sequence preceding the command sequence including the data request.

37. An article of manufacture including prestage commands to prestage data into cache from a storage system in preparation for data transfer operations, wherein a first processing unit communicates data transfer operations to a second processing unit that controls access to the storage system, the article of manufacture comprising computer readable storage media including at least one computer program embedded therein that is capable of causing the second processing unit to perform:

- receiving a prestage command from the first processing unit and a data structure capable of indicating contiguous and non-contiguous addressable locations in the storage system including the data to prestage into the cache, wherein the data structure comprises a bit map data structure having bit map values for addressable locations in the storage system, wherein the bit map values corresponding to the addressable locations including the data to prestage into cache are set to one;
- prestaging into the cache the data at the addressable locations indicated in the data structure;
- receiving a data request from the first processing unit for data at the addressable locations indicated in the data structure; and
- returning to the first processing unit the requested data from the cache.

38. The article of manufacture of claim 37, wherein the data structure comprises a bit map data structure having bit map values for addressable locations in the storage system, wherein the second processing unit prestages into cache data at the addressable locations having a corresponding bit map value of one.

39. The article of manufacture of claim 37, wherein the addressable locations in the data structure correspond to data having a logical sequential ordering within the first processing unit.

40. The article of manufacture of claim 39, wherein the addressable locations in the storage system including the data having the logical sequential ordering are at non-contiguous addressable locations in the storage system.

41. The article of manufacture of claim 37, wherein the storage system storage space is logically divided into multiple tracks, wherein each track includes one or more data records, wherein each data record includes an index area providing index information on the content of the data record and a user data area including user data, wherein the addressable locations indicated in the data structure comprise tracks in the storage system including the data records to prestage into the cache.

42. The article of manufacture of claim 37, wherein the data request from the first processing unit is for data that was prestaged into cache in a command sequence preceding the command sequence including the data request.

43. A computer readable memory device accessible to a processing unit, wherein the memory device includes a prestage command and a data structure capable of indicating contiguous and non-contiguous addressable locations in a storage system including data to prestage into a cache from the storage system in preparation for data transfer operations between a first processing unit and second processing unit, wherein the data structure comprises a bit map data structure having bit map values for addressable locations in the

21

storage system, wherein the bit map values corresponding to the addressable locations including the data to prestage into cache are set to one, wherein the prestage command and the data structure are communicated from the first processing unit to the second processing unit, and wherein the prestage command is capable of causing the second processing unit to prestage into the cache the data at the addressable locations indicated in the data structure.

44. The memory device of claim 43, wherein the data structure comprises a bit map data structure having bit map values for addressable locations in the storage system, wherein the second processing unit prestages into cache data at the addressable locations having a corresponding bit map value of one.

45. The memory device of claim 43, wherein the addressable locations in the data structure correspond to data having a logical sequential ordering within the first processing unit.

46. The memory device of claim 45, wherein the addressable locations in the storage system including the data having the logical sequential ordering are at non-contiguous addressable locations in the storage system.

47. An article of manufacture including prestage commands to prestage data into cache from a storage system in preparation for data transfer operations, wherein the storage system storage space is logically divided into multiple tracks, wherein each track includes one or more data records, wherein each data record includes an index area providing index information on the content of the data record and a user data area including user data, wherein a first processing unit communicates data transfer operations to a second processing unit that controls access to the storage system, the article of manufacture comprising computer readable storage media including at least one computer program embedded therein that is capable of causing the first processing unit to perform:

determining addressable locations in the storage system of data to prestage into cache;

generating a data structure capable of indicating contiguous and non-contiguous addressable locations in the storage system including the data to prestage into the cache by:

(i) generating a bit map data structure having bit map values for addressable locations in the storage system; and

22

(ii) setting to one the bit map values corresponding to the addressable locations including the data to prestage into cache;

transmitting a prestage command to the second processing unit which controls access to the storage system, wherein the prestage command causes the second processing unit to prestage into cache the data at the addressable locations indicated in the data structure; and

requesting data at the addressable locations indicated in the data structure, wherein the second processing unit returns the requested data from the cache.

48. An article of manufacture including prestage commands to prestage data into cache from a storage system in preparation for data transfer operations, wherein the storage system storage space is logically divided into multiple tracks, wherein each track includes one or more data records, wherein each data record includes an index area providing index information on the content of the data record and a user data area including user data, wherein a first processing unit communicates data transfer operations to a second processing unit that controls access to the storage system, the article of manufacture comprising computer readable storage media including at least one computer program embedded therein that is capable of causing the second processing unit to perform:

receiving a prestage command from the first processing unit and a data structure capable of indicating contiguous and non-contiguous addressable locations in the storage system including the data to prestage into the cache, wherein the data structure comprises a bit map data structure having bit map values for addressable locations in the storage system, wherein the bit map values corresponding to the addressable locations including the data to prestage into cache are set to one; prestaging into the cache the tracks at the addressable locations indicated in the data structure;

receiving a data request from the first processing unit for data at the addressable locations indicated in the data structure; and

returning to the first processing unit the requested data from the cache.

* * * * *